

Vorlesung  
Internet Algorithmen

SS 2005  
Prof.Dr. Georg Schnitger

**Vorläufige Version**

19. Februar 2009



# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>5</b>
1.1	Notation . . . . .	6
1.2	Wichtige Ungleichungen . . . . .	7
1.3	Graphen . . . . .	8
1.4	Grundlagen aus der Stochastik . . . . .	10
1.5	Lineare Programmierung . . . . .	19
<b>I</b>	<b>Die Bearbeitung großer Datenmengen</b>	<b>23</b>
<b>2</b>	<b>Suchmaschinen</b>	<b>25</b>
2.1	Die Architektur von Suchmaschinen . . . . .	25
2.2	Google . . . . .	26
2.2.1	Inhalt-abhängiger Page-Rank . . . . .	29
2.3	Hubs und Authorities . . . . .	30
2.3.1	Inhalt-abhängige Suche . . . . .	33
2.4	Meta-Suchmaschinen . . . . .	34
2.4.1	Der Unmöglichkeitssatz von Arrow . . . . .	35
2.4.2	Die Kendall-Distanz . . . . .	37
2.5	Das Semantische Netz . . . . .	40
2.6	Zusammenfassung . . . . .	41
<b>3</b>	<b>Hashing für Web-Anwendungen</b>	<b>43</b>
3.1	Verteiltes Hashing in Peer-to-Peer Netzwerken . . . . .	43
3.1.0.1	Chord . . . . .	44
3.2	Ähnlichkeitsbestimmung . . . . .	46
3.2.1	Min-Hashing . . . . .	48
3.2.2	Wann existieren ähnlichkeitsbewahrende Hashfunktionen? . . . . .	49
3.3	Bloom-Filter . . . . .	51
3.3.1	Anwendungen . . . . .	54
3.4	Zusammenfassung . . . . .	55
<b>4</b>	<b>Streaming Data</b>	<b>57</b>
4.1	Stichproben . . . . .	58
4.2	Häufigkeitsmomente . . . . .	62
4.2.1	Streaming-Data und Kommunikation . . . . .	62
4.2.2	Die Bestimmung von $H_0$ . . . . .	65

4.2.3	Die Bestimmung von $H_2$ . . . . .	69
4.3	Häufigkeitsanfragen . . . . .	71
4.3.1	Heavy Hitters . . . . .	71
4.3.2	Der Bloom-Filter Sketch . . . . .	73
4.3.3	Häufigkeitseigenschaften . . . . .	76
4.4	Algorithmen für Zeitfenster: Zählen der letzten Einsen . . . . .	78
4.5	Histogramme . . . . .	79
<b>5</b>	<b>Queueing-Strategien</b>	<b>85</b>
5.1	Stabilität . . . . .	86
5.2	Instabile Queueing-Strategien . . . . .	90
5.3	Queuegröße . . . . .	94
5.4	Eine randomisierte Queueing-Strategie . . . . .	97
<b>6</b>	<b>Erasur Codes</b>	<b>101</b>
6.1	Informationsrate . . . . .	105
6.2	Berechnung der Grad-Sequenzen . . . . .	105
6.3	Leistungsdaten der Tornado Codes . . . . .	109
<b>7</b>	<b>End-to-End Congestion Control</b>	<b>113</b>
7.1	AIMD für eine Ressource . . . . .	117
7.2	Alternative Geschwindigkeitsregeln . . . . .	121
<b>8</b>	<b>IP-Traceback</b>	<b>125</b>
	<b>Literaturverzeichnis</b>	<b>131</b>

# Kapitel 1

## Einführung

Das Skript ist in die drei folgenden Themengebiete unterteilt:

- I Bearbeitung großer Datenmengen,
- II Internet Routing und
- III algorithmische Spieltheorie.

Im ersten Teil betrachten wir zuerst Suchmaschinen und beschreiben das Page-Rank-Verfahren von Google sowie den Hubs-Authorities Ansatz von Kleinberg. Im Kapitel 3 untersuchen wir verteilte Hashing-Verfahren, ähnlichkeitsbewahrendes Hashing sowie speicherplatz-effizientes Hashing und wenden die entsprechenden Methoden auf besonders Daten-reiche Probleme im verteilten Rechnen an. In Kapitel 4 untersuchen wir das *Streaming Data* Modell: Große Datenmengen sind in Echtzeit zu bewältigen, wobei wir annehmen müssen, dass uns nur ein einziger Datendurchlauf zur Verfügung steht. Wir entwickeln algorithmische Lösungen vorwiegend für die statistische Datenanalyse, und zeigen in vielen Fällen, dass unsere Lösungen (fast-)optimal sind.

Der zweite Teil befasst sich mit dem Internet Routing. Nach einer kurzen Einführung in das Internet Protokoll (IP), das Transmission Control Protokoll (TCP) sowie in das Intra-Domain und Inter-Domain Routing beginnen wir mit einer Untersuchung von Queueing Strategien für Router in Abschnitt 5. Nach dem *Unicast* Routing Problem, also dem Versenden einer Nachricht von einem Sender an einen Empfänger, besprechen wir auch kurz das *Multicast* Routing Problem, in dem ein multimedialer Datenstrom (Video on Demand, Interaktives Fernsehen, Live Übertragungen) von einem Sender an mehrere Empfänger zu verschicken ist. Wir stellen den Tornado Code detailliert vor, der durch eine redundante Daten-Kodierung in Echtzeit die Netzbelastung wesentlich reduziert.

In Abschnitt 7 beschreiben und analysieren wir die *Additive Increase, Multiplicative Decrease* Strategie, mit der TCP die Emissionsrate eines Senders der Netzbelastung anpasst, und skizzieren alternative Methoden der Verkehrssteuerung im Internet. Wir schließen in Abschnitt 8 mit einer Betrachtung von Verfahren zur Bekämpfung von Denial-of-Service Attacken.

Das Internet wird von einer Vielzahl von Personen und Institutionen betrieben und genutzt, die angetrieben von eigenen, meist wirtschaftlichen Interessen miteinander agieren, Koalitionen bilden, aber auch konkurrieren. Daher scheint es naiv zu erwarten, dass Ressourcen gerecht geteilt werden und dass sich Rechner stets an vorgegebene Protokolle halten. Wir betrachten

deshalb im dritten und letzten Teil die algorithmische Spieltheorie, um den Egoismus-Aspekt der beteiligten Akteure (oder Spieler) untersuchen zu können. Wir beschränken uns auf die nicht-kooperativen Spieltheorie und nehmen an, dass die Spieler miteinander konkurrieren ohne Koalitionen bilden zu können.

Nash-Gleichgewichte beschreiben Spielkonfigurationen, die im gewissen Sinne für alle Spieler profitabel sind. Wesentliche Fragestellungen in Kapitel ?? betreffen die Existenz von Nash-Gleichgewichten (Abschnitte ?? und ??) sowie ihre Berechnung (Abschnitt ??). Die Berechnung optimaler Spielstrategien verlangt Spieler mit unbeschränkter Rechenkraft, und wir untersuchen deshalb in Abschnitt ?? die Konsequenz beschränkter Rechenkraft.

Dann wenden wir den Begriff des Nash-Gleichgewichts an, um verschiedene Varianten von TCP in Kapitel ?? zu evaluieren und um den Effekt des egoistischen Routings im Kapitel ?? zu untersuchen. Wir schließen in Kapitel ?? mit dem Entwurf von Mechanismen, also dem Entwurf von Spielregeln (Gebühren und/oder Anreizen), damit sich rationale egoistische Spieler wie sozial erwünscht verhalten.

## 1.1 Notation

$\mathbb{R}$  bezeichnet die Menge der reellen Zahlen,  $\mathbb{R}_{\geq 0}$ , die Menge der nicht-negativen reellen Zahlen,  $\mathbb{Q}$  die Menge der rationalen Zahlen und  $\mathbb{N}$  die Menge der natürlichen Zahlen.

Für eine reelle Zahl  $x$  ist  $\lfloor x \rfloor$  die größte natürliche Zahl kleiner oder gleich  $x$  und  $\lceil x \rceil$  die kleinste natürliche Zahl größer oder gleich  $x$ .

Wir benutzen die asymptotische Notation, um den Ressourcenbedarf für große Eingabelängen zu analysieren. Für zwei Funktionen  $f, g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$  schreiben wir

- (a)  $\mathbf{f} = \mathbf{O}(\mathbf{g})$ , falls  $f(n) \leq cg(n)$  für alle  $n \geq N$ , wobei  $N$  eine ausreichend große natürliche Zahl und  $c > 0$  eine passend gewählte Konstante ist.
- (b)  $\mathbf{f} = \mathbf{\Omega}(\mathbf{g})$ , falls  $g = O(f)$ .
- (c)  $\mathbf{f} = \mathbf{\Theta}(\mathbf{g})$ , falls  $f = O(g)$  und  $g = O(f)$ .
- (d)  $\mathbf{f} = \mathbf{o}(\mathbf{g})$ , falls  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ .
- (e)  $\mathbf{f} \sim \mathbf{g}$ , falls  $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$ .

Das innere Produkt  $\langle x, y \rangle$  von zwei Vektoren  $x = (x_1, \dots, x_n)$  and  $y = (y_1, \dots, y_n)$  in  $\mathbb{R}^n$  wird durch

$$\langle x, y \rangle = x_1 \cdot y_1 + \dots + x_n \cdot y_n.$$

definiert und die Euklid'sche Norm  $\|x\|$  durch  $\|x\| = \sqrt{\langle x, x \rangle}$ .

Eine reell-wertige Funktion  $f(x)$  ist genau dann konvex (bzw. konkav), wenn

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y) \text{ (resp. } f(\lambda x + (1 - \lambda)y) \geq \lambda f(x) + (1 - \lambda)f(y))$$

für jedes  $0 \leq \lambda \leq 1$  gilt. Die Konvexität von  $f$  bedeutet geometrisch gesehen, dass eine Gerade mit den Endpunkten  $(x, f(x))$  und  $(y, f(y))$  stets oberhalb der Kurve liegt. Deshalb ist es für die Konvexität ausreichend, wenn die zweite Ableitung nicht-negativ ist.

## 1.2 Wichtige Ungleichungen

Die Ungleichung von Cauchy-Schwartz besagt, dass das innere Produkt von zwei Vektoren der Länge Eins durch Eins beschränkt ist.

**Lemma 1.1** *Es seien  $x$  und  $y$  Vektoren im  $\mathbb{R}^n$ . Dann gilt*

$$\langle x, y \rangle \leq \|x\| \cdot \|y\|.$$

**Beweis:** Wir betrachten das innere Produkt  $\langle u, v \rangle$  von zwei Vektoren  $u$  und  $v$  der Norm 1. Der Wert des inneren Produkts stimmt mit dem Kosinus des Winkels zwischen  $u$  und  $v$  überein und ist deshalb durch 1 nach oben beschränkt. Also folgt  $\langle \frac{x}{\|x\|}, \frac{y}{\|y\|} \rangle \leq 1$  und damit die Behauptung.  $\square$

Wir benötigen auch eine Abschätzung von  $1 \pm x$  durch die e-Funktion.

**Lemma 1.2** *Für jedes  $x > -1$  gilt*

$$e^{x/(1+x)} \leq 1 + x \leq e^x.$$

*Darüberhinaus gilt  $1 + x \leq e^x$  für alle  $x \in \mathbb{R}$ .*

**Beweis:** Wir beschränken uns zuerst auf den Fall  $x \geq 0$ . Der natürliche Logarithmus ist eine konkave Funktion und deshalb ist die Steigung der Sekante in den Punkten 1 und  $1+x$  durch die Tangentensteigungen in den Punkten 1 nach oben und  $1+x$  nach unten beschränkt. Dies liefert die Ungleichung

$$\frac{1}{1+x} \leq \frac{\ln(1+x) - \ln(1)}{(1+x) - 1} = \frac{\ln(1+x)}{x} \leq 1 \quad (1.1)$$

oder äquivalent

$$\frac{x}{1+x} \leq \ln(1+x) \leq x. \quad (1.2)$$

Die erste Behauptung folgt also für  $x \geq 0$  durch Exponentieren. Für  $x \in ]-1, 0]$  brauchen wir nur zu beachten, dass diesmal die Ungleichung

$$1 \leq \frac{\ln(1+x) - \ln(1)}{(1+x) - 1} \leq \frac{1}{1+x}$$

gilt: Das Argument ist analog zur Ableitung von Ungleichung (1.1). Wir erhalten jetzt wieder Ungleichung (1.2), und damit die Behauptung, wenn wir (1.1) mit der negativen Zahl  $x$  multiplizieren.

Damit gilt aber auch  $1 + x \leq e^x$  für alle reellen Zahlen, da  $1 + x$  für  $x \leq -1$  negativ und  $e^x$  stets positiv ist.  $\square$

Wie verhält sich die ‘‘Konvexitätsungleichung’’, wenn wir eine konvexe Funktion auf eine Summe von mehr als zwei Termen anwenden?

**Lemma 1.3 (Jensen’s Ungleichung)** *Wenn  $0 \leq \lambda_i \leq 1$ ,  $\sum_{i=1}^r \lambda_i = 1$  und wenn  $f$  konvex ist, dann gilt*

$$f\left(\sum_{i=1}^r \lambda_i x_i\right) \leq \sum_{i=1}^r \lambda_i f(x_i).$$

**Beweis:** Wir führen eine Induktion nach der Anzahl  $r$  der Terme. Die Basis  $r = 2$  ist offensichtlich richtig, da  $f$  konvex ist. Für den Induktionsschritt von  $r$  auf  $r + 1$  ersetzen wir die Summe der beiden Terme  $\lambda_1 x_1 + \lambda_2 x_2$  durch den einen Term

$$(\lambda_1 + \lambda_2) \left( \frac{\lambda_1}{\lambda_1 + \lambda_2} x_1 + \frac{\lambda_2}{\lambda_1 + \lambda_2} x_2 \right) = (\lambda_1 + \lambda_2) \cdot x'_2$$

und können dann die Induktionshypothese auf  $x'_2, x_3, \dots, x_n$  anwenden.  $\square$

Wir erhalten jetzt sofort eine wichtige Beziehung zwischen dem arithmetischen und dem geometrischen Mittel.

**Lemma 1.4**  $a_1, \dots, a_n$  seien nicht-negative Zahlen. Dann gilt

$$\frac{1}{n} \sum_{i=1}^n a_i \geq \left( \prod_{i=1}^n a_i \right)^{1/n}. \quad (1.3)$$

**Beweis:** Wir setzen  $f(x) = 2^x$ ,  $\lambda_1 = \dots = \lambda_n = 1/n$  und  $x_i = \log_2 a_i$ , für alle  $i = 1, \dots, n$ . Nach Jensen's Ungleichung folgt

$$\frac{1}{n} \sum_{i=1}^n a_i = \sum_{i=1}^n \lambda_i f(x_i) \geq f \left( \sum_{i=1}^n \lambda_i x_i \right) = 2^{(\sum_{i=1}^n x_i)/n} = \left( \prod_{i=1}^n a_i \right)^{1/n}.$$

$\square$

Die beiden letzten Ungleichung sind für die Abschätzung von Binomialkoeffizienten wichtig.

**Lemma 1.5** (a) Die Stirling Formel besagt

$$n! = \sqrt{2\pi \cdot n} \cdot \left( \frac{n}{e} \right)^n \cdot \left( 1 + \frac{1}{12 \cdot n} + O\left(\frac{1}{n^2}\right) \right).$$

(b) Es ist

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} \leq \left( \frac{n \cdot e}{k} \right)^k.$$

## 1.3 Graphen

Ein *ungerichteter Graph* ist ein Paar  $G = (V, E)$ , das aus einer Knotenmenge  $V$  und einer Kantenmenge  $E$  besteht, wobei alle Kanten Teilmengen von  $V$  der Größe zwei sind. Wir sagen, dass ein Knoten  $v$  mit der Kante  $e$  inzident ist, wenn  $v \in e$ . Die beiden mit einer Kante  $e$  inzidenten Knoten sind die Endpunkte von  $E$ . Zwei Knoten  $u, v \in V$  heißen *adjazent*, oder *benachbart*, wenn  $\{u, v\}$  eine Kante von  $G$  ist. In einem *gerichteten Graphen* sind Kanten geordnete Paare, d.h. es gilt stets  $E \subseteq V^2$ , und wir haben die Möglichkeit den Anfangs- und Endpunkt einer Kante auszuzeichnen. Der Knoten  $v$  heißt ein Nachbar (oder Nachfolger) von  $u$ , falls  $(u, v) \in E$ .

Die Anzahl der Nachbarn eines Knotens  $u$  ist der *Grad* von  $u$ . Der Grad eines Graphen ist der maximale Grad eines Knotens.

Ein *Spaziergang* der Länge  $k$  in  $G$  ist eine Folge  $v_0, e_1, v_1, \dots, e_k, v_k$  von Knoten und Kanten, so dass  $e_i = \{v_{i-1}, v_i\} \in E$ , bzw.  $(v_{i-1}, v_i) \in E$ . Ein Spaziergang ohne wiederholte Knoten

ist ein *Weg*. Ein *Kreis* der Länge  $k$  ist ein Spaziergang  $v_0, \dots, v_k$  der Länge  $k$  mit  $v_0 = v_k$  und der Eigenschaft, dass  $v_0, \dots, v_{k-1}$  ein Weg ist. Die *Distanz* zwischen zwei Knoten ist die minimale Länge eines Weges zwischen den beiden Knoten.

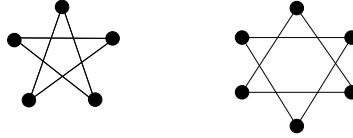


Abbildung 1.1: Der erste Graph ist ein Kreis der Länge 5. Der zweite Graph besteht aus zwei Kreisen der Länge 3.

Ein *Hamilton-Kreis* ist ein Kreis, der jeden Knoten genau einmal durchläuft. Ein *Euler-Kreis* ist ein Spaziergang, der zu seinem Anfangsknoten zurückkehrt und jede Kante genau einmal durchläuft.

Eine Knotenmenge in einem ungerichteten Graphen ist *zusammenhängend*, wenn je zwei Knoten der Menge durch einen Weg verbunden sind. Eine *Zusammenhangskomponente* ist eine größte zusammenhängende Knotenmenge.

Ein *Wald* ist ein ungerichteter Graph ohne Kreise. Ein *Baum* ist ein zusammenhängender Wald. (Mit anderen Worten, ein Wald ist eine knoten-disjunkte Vereinigung von Bäumen.)

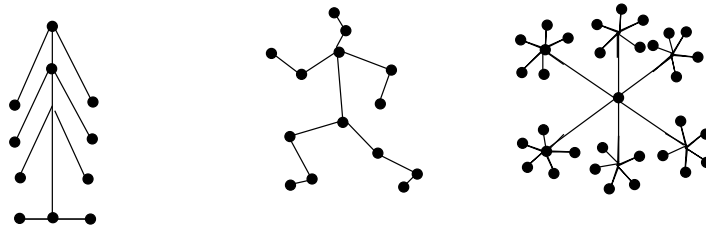


Abbildung 1.2: Ein Wald, der aus drei Bäumen besteht.

In einem Baum mit *Wurzel* wird genau ein Knoten als Wurzel ausgezeichnet. Wenn der Weg von der Wurzel zum Knoten  $w$  die Kante  $\{v, w\}$  benutzt, dann heißt  $w$  ein *Kind* von  $v$  und  $v$  der *Vater* von  $w$ . Knoten vom Grad 1 heißen *Blätter*. Die *Tiefe* eines Knotens ist die Länge des Weges von der Wurzel bis zu dem Knoten. Die Tiefe eines Baums ist die größte Tiefe eines Blatts. Ein Baum heißt *binär*, wenn die Wurzel den Grad zwei und jedes Nicht-Blatt den Grad drei besitzt. Ein *vollständiger binärer* Baum der Tiefe  $T$  ist ein binärer Baum, in dem alle Blätter die Tiefe  $T$  besitzen.

Ein *vollständiger Graph* oder *eine Clique* ist ein Graph, in dem je zwei Knoten adjazent sind. Eine *unabhängige Menge* ist eine Knotenmenge, in der je zwei Knoten **nicht** durch eine Kante verbunden sind. Ein Graph ist *bipartit*, wenn seine Knotenmenge in zwei unabhängige Mengen zerlegt werden kann.

Eine legale *Färbung* eines ungerichteten Graphen  $G = (V, E)$  ist eine Farbzueweisung an die Knoten, so dass adjazente Knoten verschiedene Farben erhalten. Eine legale Färbung zerlegt somit die Knotenmenge in unabhängige Mengen. Die minimale Farbenzahl, die für die Färbung eines Graphen  $G$  benötigt wird, heißt die *chromatische Zahl*  $\chi(G)$  von  $G$ .

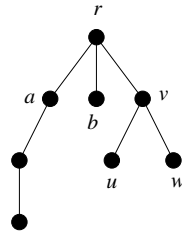


Abbildung 1.3: Ein Baum mit Wurzel der Tiefe 3;  $u$  und  $w$  sind Kinder des Vaters  $v$ .

### Aufgabe 1

(a) Zeige, dass die Tiefe eines binären Baums mit  $n$  Blättern mindestens  $\lceil \log_2 n \rceil$  beträgt.

(b) Sei  $B$  ein binärer Baum mit  $N$  Blättern und sei  $T_i$  die Tiefe des  $i$ ten Blatts. Zeige die sogenannte Kraft'sche Ungleichung

$$\sum_{i=1}^n 2^{-T_i} \leq 1.$$

Hinweis: Assoziiere einen binären String mit jedem Blatt.

---

## 1.4 Grundlagen aus der Stochastik

Ein *endlicher Wahrscheinlichkeitsraum* besteht aus einer endlichen Menge  $\Omega$  (dem *Stichprobenraum*) und einer Funktion

$$\text{prob} : \Omega \rightarrow [0, 1]$$

(der *Wahrscheinlichkeitsverteilung*), so dass  $\sum_{x \in \Omega} \text{prob}[x] = 1$ . Der Wahrscheinlichkeitsraum repräsentiert das Zufallsexperiment, ein Element von  $\Omega$  auszuwählen und  $\text{prob}[x]$  ist die Wahrscheinlichkeit, dass  $x$  gewählt wird.

Die Elemente  $x \in \Omega$  heißen *Stichproben* und Teilmengen  $A \subseteq \Omega$  heißen *Ereignisse*. Die Wahrscheinlichkeit eines Ereignisses  $A$  ist

$$\text{prob}[A] = \sum_{x \in A} \text{prob}[x],$$

also die Wahrscheinlichkeit, dass ein Element mit Eigenschaft  $A$  gewählt wird. Einige elementare Eigenschaften folgen direkt aus diesen Definitionen. Für zwei Ereignisse  $A$  and  $B$  (und das Komplement  $\bar{A} = \Omega \setminus A$ ) gilt

$$(1) \text{prob}[A \cup B] = \text{prob}[A] + \text{prob}[B] - \text{prob}[A \cap B];$$

$$(2) \text{prob}[\bar{A}] = 1 - \text{prob}[A];$$

$$(3) \text{prob}[A \cap B] \geq \text{prob}[A] - \text{prob}[\bar{B}];$$

$$(4) \text{ Wenn } B_1, \dots, B_m \text{ eine Zerlegung von } \Omega \text{ ist, dann ist } \text{prob}[A] = \sum_{i=1}^m \text{prob}[A \cap B_i].$$

$$(5) \text{prob}[\bigcup_{i=1}^n A_i] \leq \sum_{i=1}^n \text{prob}[A_i].$$

Besonders Eigenschaft (5) werden wir häufig benutzen. Für zwei Ereignisse  $A$  und  $B$  ist  $\text{prob}[A|B]$  die *bedingte Wahrscheinlichkeit* von  $A$  gegeben  $B$ , also die Wahrscheinlichkeit,

dass Ereignis  $A$  eintritt, wenn man bereits weiß, dass Ereignis  $B$  eingetreten ist. Formal definieren wir

$$\text{prob}[A|B] = \frac{\text{prob}[A \cap B]}{\text{prob}[B]},$$

falls  $\text{prob}[B] \neq 0$ . Wenn wir zum Beispiel eine Zahl aus der Menge  $\{1, \dots, 6\}$  zufällig ziehen und wenn  $A$  das Ereignis ist, dass 2 die gezogene Zahl ist und  $B$  das Ereignis ist, dass die gezogene Zahl gerade ist, dann ist  $\text{prob}[A|B] = 1/3$ , wohingegen  $\text{prob}[B|A] = 1$  gilt.

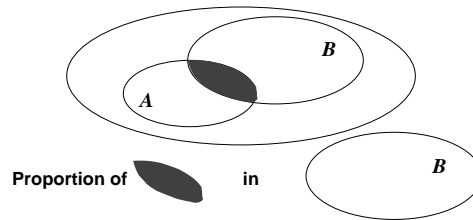


Abbildung 1.4: Interpretation von  $\text{prob}[A|B]$ , wenn alle Elemente gleichwahrscheinlich sind.

Zwei Ereignisse  $A$  und  $B$  heißen *unabhängig*, wenn  $\text{prob}[A \cap B] = \text{prob}[A] \cdot \text{prob}[B]$ . Beachte, dass diese Bedingung äquivalent mit  $\text{prob}[A|B] = \text{prob}[A]$  wie auch mit  $\text{prob}[B|A] = \text{prob}[B]$  ist. Wenn alle Elemente gleichwahrscheinlich sind, dann sind zwei Ereignisse genau dann unabhängig, wenn der Anteil  $|A|/|\Omega|$  von Ereignis  $A$  im Stichprobenraum mit dem Anteil  $|A \cap B|/|B|$  des Ereignisses  $A \cap B$  im Ereignis  $B$  übereinstimmt.

Beachte, dass Unabhängigkeit nichts mit der Disjunktheit von Ereignissen zu tun hat: Wenn zum Beispiel  $\text{prob}[A] > 0$ ,  $\text{prob}[B] > 0$  und  $A \cap B = \emptyset$  gilt, dann sind die Ereignisse  $A$  und  $B$  abhängig!

**Beispiel 1.1** Wir beschreiben das „Monty Hall Problem“. In einer Game Show ist hinter einer von drei Türen ein Preis verborgen. Ein Zuschauer rät eine der drei Türen und der Showmaster Monty Hall wird daraufhin eine weitere Tür öffnen, hinter der sich aber kein Preis verbirgt. Der Zuschauer erhält jetzt die Möglichkeit, seine Wahl zu ändern. Sollte er dies tun? Wir betrachten die Ereignisse

- $P_i$ , dass sich der Preis hinter Tür  $i$  befindet,
- $Z_i$ , dass der Zuschauer zuerst Tür  $i$  wählt und
- $M_i$ , dass Monty Hall Tür  $i$  nach der ersten Wahl des Zuschauers öffnet.

Wir nehmen o.B.d.A. an, dass der Zuschauer zuerst Tür 1 wählt und dass Monty Hall daraufhin Tür 2 öffnet; desweiteren nehmen wir an, dass der Zuschauer wie auch Monty Hall seine Wahl jeweils nach der Gleichverteilung trifft. Wir müssen die bedingten Wahrscheinlichkeiten  $\text{prob}[P_1 | Z_1, M_2]$  und  $\text{prob}[P_3 | Z_1, M_2]$  berechnen und beachten, dass  $\text{prob}[P_1 | Z_1, M_2] + \text{prob}[P_3 | Z_1, M_2] = 1$  gilt, denn der Preis befindet sich nicht hinter der geöffneten Tür 2. Nach Definition der bedingten Wahrscheinlichkeiten ist

$$\begin{aligned} \text{prob}[P_1 | Z_1, M_2] \cdot \text{prob}[Z_1, M_2] &= \text{prob}[Z_1, M_2 | P_1] \cdot \text{prob}[P_1] \quad \text{und} \\ \text{prob}[P_3 | Z_1, M_2] \cdot \text{prob}[Z_1, M_2] &= \text{prob}[Z_1, M_2 | P_3] \cdot \text{prob}[P_3]. \end{aligned}$$

Wir bestimmen jeweils die rechten Seiten und erhalten

$$\text{prob}[P_1 | Z_1, M_2] \cdot \text{prob}[Z_1, M_2] = \left(\frac{1}{3} \cdot \frac{1}{2}\right) \cdot \frac{1}{3},$$

denn Monty Hall kann die Türen 2 und 3 öffnen. Andererseits ist

$$\text{prob}[P_3 | Z_1, M_2] \cdot \text{prob}[Z_1, M_2] = \left(\frac{1}{3} \cdot 1\right) \cdot \frac{1}{3},$$

denn Monty Hall kann nur Tür 2 öffnen. Also ist

$$\text{prob}[P_3 | Z_1, M_2] \cdot \text{prob}[Z_1, M_2] = 2 \cdot \text{prob}[P_1 | Z_1, M_2] \cdot \text{prob}[Z_1, M_2]$$

und wir erhalten  $\text{prob}[P_3 | Z_1, M_2] = \frac{2}{3}$  und  $\text{prob}[P_1 | Z_1, M_2] = \frac{1}{3}$ : Der Zuschauer sollte seine Wahl stets ändern!

Eine *Zufallsvariable* ist eine Funktion  $X : \Omega \rightarrow \mathbb{R}$ . Wenn zum Beispiel  $X$  eine aus der Menge  $\{1, \dots, n\}$  zufällig gezogene Zahl ist, dann sind  $X$ ,  $Y = 2X$  und  $Z =$  „die Anzahl der Primteiler von  $X$ “ Zufallsvariablen. Eine *Indikatorvariable* für das Ereignis  $A$  ist eine 0-1 wertige Zufallsvariable mit

$$X(\omega) = \begin{cases} 1 & \text{wenn } \omega \in A; \\ 0 & \text{sonst.} \end{cases}$$

**Beispiel 1.2** Die Lösung des Monty Hall Problems kann mit Hilfe von Zufallsvariablen kompakt erklärt werden: Sei  $W$  die *Zufallsvariable*, die die erste Wahl des Zuschauers beschreibt und sei  $T$  die Zufallsvariable, die die richtige Tür beschreibt. Dann findet die Änderungsstrategie genau dann die richtige Tür, wenn  $W$  und  $T$  unterschiedliche Werte annehmen und dieses *Ereignis* hat die Wahrscheinlichkeit  $\frac{2}{3}$ . Fazit: Mit höherer Wahrscheinlichkeit war die erste Wahl schlecht und wird durch die veränderte Wahl richtig.

Zufallsvariablen  $X_1, \dots, X_n$  heißen genau dann *unabhängig*, wenn für alle  $x_1, \dots, x_n \in \mathbb{R}$

$$\text{prob}[X_1 = x_1 \wedge \dots \wedge X_n = x_n] = \prod_{i=1}^n \text{prob}[X_i = x_i]$$

gilt. Die Zufallsvariablen heißen *k-fach unabhängig*, wenn  $X_{i_1}, \dots, X_{i_k}$  für jede Teilmenge  $\{i_1, \dots, i_k\} \subseteq \{1, \dots, n\}$  von  $k$  Elementen unabhängig ist.

Beachte, dass wiederholte Experimente unabhängigen Experimenten entsprechen. Desweiteren sind unabhängige Zufallsvariablen  $X_1, \dots, X_n$  *k-fach unabhängig* für jedes  $k < n$ . Die Umkehrung dieser Aussage gilt nicht.

---

### Aufgabe 2

Konstruiere paarweise unabhängige Zufallsvariablen  $X_1, X_2, X_3$ , die aber nicht unabhängig sind.

---

Der *Erwartungswert* einer Zufallsvariable kann für jede reell-wertige Zufallsvariable  $X$  definiert werden und ist, intuitiv gesprochen, der zu „erwartende“ Wert, wenn wir das Zufallsexperiment  $X$  mehrfach wiederholen und das durchschnittliche Ergebnis berechnen. Formal definieren wir den Erwartungswert von  $X$  als die Summe

$$E[X] = \sum_i i \cdot \text{prob}[X = i]$$

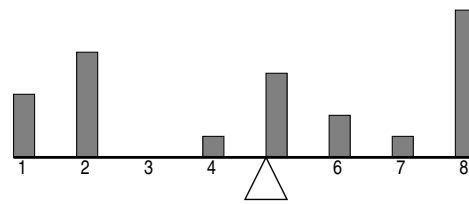


Abbildung 1.5: Interpretation des Erwartungswerts als „Masse-Zentrum“: Die Stichproben haben das Gewicht  $p_i = \text{prob}[X = a_i]$  auf der  $x$ -Achse an den Stellen  $a_i$  für  $i = 1, \dots, n$ .

über alle Werte  $i$  des Wertebereichs: Wenn wir zum Beispiel würfeln, dann ist  $E[X] = \sum_{i=1}^6 \frac{i}{6} = 3.5$  der Erwartungswert. (An diesem Beispiel können wir beobachten, dass der Erwartungswert von  $X$  kein Element des Wertebereichs von  $X$  sein muss!)

Erwarte nie das „Erwartete“! Und tatsächlich sagt der Erwartungswert in vielen ärgerlichen Fällen nicht viel über das Verhalten der Zufallsvariable  $X$  aus. Der Erwartungswert ist nämlich nur dann vertrauenswürdig, wenn man zeigen kann, dass größere Abweichungen vom Erwartungswert unwahrscheinlich sind. Die „Distanz“ zwischen einer Zufallsvariable  $X$  und ihrem Erwartungswert misst man zum Beispiel mit der *Varianz*, die durch

$$\text{Var}[X] = E[(X - E[X])^2].$$

definiert ist. Eine sehr wichtige Eigenschaft des Erwartungswerts ist seine *Linearität*.

**Satz 1.6 (Linearität des Erwartungswerts.)** Für Zufallsvariablen  $X, Y$  und reelle Zahlen  $a, b$  gilt

$$E[a \cdot X + b \cdot Y] = a \cdot E[X] + b \cdot E[Y].$$

**Beweis:**  $x_1, \dots, x_n$  und  $y_1, \dots, y_m$  seien Elemente des Wertebereichs von  $X$ , bzw.  $Y$ . Da für jede reelle Zahl  $a$

$$E[a \cdot X] = \sum_{i=1}^n a \cdot x_i \cdot \text{prob}[X = x_i] = a \cdot \sum_{i=1}^n x_i \cdot \text{prob}[X = x_i] = a \cdot E[X],$$

gilt, genügt der Nachweis von  $E[X + Y] = E[X] + E[Y]$ . Da die Ereignisse  $Y = y_j$  für verschiedene  $y_j$ 's disjunkt sind, folgt  $\text{prob}[X = x_i] = \sum_{j=1}^m \text{prob}[X = x_i, Y = y_j]$ , und ein ähnliches Ergebnis gilt natürlich für  $\text{prob}[Y = y_j]$ . Deshalb folgt

$$\begin{aligned} E[X + Y] &= \sum_{i=1}^n \sum_{j=1}^m (x_i + y_j) \text{prob}[X = x_i, Y = y_j] \\ &= \sum_{i=1}^n \sum_{j=1}^m x_i \text{prob}[X = x_i, Y = y_j] + \sum_{j=1}^m \sum_{i=1}^n y_j \text{prob}[X = x_i, Y = y_j] \\ &= \sum_{i=1}^n x_i \text{prob}[X = x_i] + \sum_{j=1}^m y_j \text{prob}[Y = y_j] \\ &= E[X] + E[Y]. \end{aligned}$$

□

- (a) Zeige, dass die Varianz mit der Formel  $\text{Var}[X] = \text{E}[X^2] - (\text{E}[X])^2$  berechnet werden kann.
- (b) Zeige, dass stets  $\text{Var}[a \cdot X + b] = a^2 \cdot \text{Var}[X]$  gilt.
- (c)  $X$  und  $Y$  seien unabhängige Zufallsvariablen. Zeige, dass  $\text{E}[X \cdot Y] = \text{E}[X] \cdot \text{E}[Y]$  and  $\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y]$  gilt.
- (d)  $X$  sei eine Zufallsvariable, die nur natürliche Zahlen als Werte annimmt. Zeige, dass  $\text{E}[X^2] \geq \text{E}[X]$ ,  $\text{prob}[X = 0] \geq 1 - \text{E}[X]$  und  $\text{E}[X] = \sum_{x=1}^{\infty} \text{prob}[X \geq x]$  gelten.
- (e) Benutze die Ungleichung von Cauchy-Schwarz, um nachzuweisen, dass  $\text{E}[X]^2 \leq \text{E}[X^2]$  für jede Zufallsvariable  $X$  gilt.
- (f) Wir haben  $n$  Briefe, die an verschiedene Personen adressiert sind und  $n$  Umschläge mit den jeweiligen Adressen. Wir stecken die Briefe zufällig und unabhängig voneinander in Umschläge und erlauben, dass mehrere Briefe in den selben Umschlag gesteckt werden. Wieviele Briefe landen im richtigen Umschlag?
- (g) Zeige, dass es für beliebige Vektoren  $v_1, \dots, v_n \in \{+1, -1\}^n$  stets Koeffizienten  $\epsilon_1, \dots, \epsilon_n \in \{+1, -1\}$  gibt, so dass

$$\|\epsilon_1 v_1 + \dots + \epsilon_n v_n\| \leq n$$

gilt.

*Hinweis:* Wähle die Koeffizienten  $\epsilon_i$  zufällig und unabhängig voneinander, wobei  $\epsilon_i = 1$  mit Wahrscheinlichkeit  $1/2$  gelte, und benutze die Linearität des Erwartungswerts, um die erwartete Länge des Vektors  $\sum \epsilon_i v_i$  zu berechnen.

**Die Methode der Erzeugendenfunktionen.** Es gibt eine allgemeine Methode, um den Erwartungswert  $\text{E}[X]$  wie auch die Varianz  $\text{Var}[X]$  für eine beliebige diskrete Zufallsvariable  $X : \Omega \rightarrow \mathbb{N}$  zu berechnen. Wir setzen  $p_k = \text{prob}[X = k]$  und definieren die *Erzeugendenfunktion* von  $X$  durch

$$F_X(x) := \sum p_k x^k,$$

wobei wir über alle Werte  $k$  des Wertebereichs von  $X$  summieren.

**Satz 1.7** *Für jede diskrete Zufallsvariable gilt*

$$F_X(1) = 1 \tag{1.4}$$

$$\text{E}[X] = F'_X(1) \tag{1.5}$$

$$\text{Var}[X] = F''_X(1) + \text{E}[X] - \text{E}[X]^2. \tag{1.6}$$

**Beweis:** Die erste Gleichung folgt sofort aus  $F_X(1) = \sum p_k = 1$ .

Wir berechnen die erste Ableitung  $F'_X(x) = \sum k p_k x^{k-1}$  und erhalten  $F'_X(1) = \sum k p_k = \text{E}[X]$ . Schließlich berechnen wir die zweite Ableitung  $F''_X(x) = \sum k(k-1) p_k x^{k-2}$  und folgern

$$F''_X(1) = \sum k(k-1) p_k = \sum k^2 p_k - \sum k p_k = \text{E}[X^2] - \text{E}[X].$$

Wenn wir also  $\text{E}[X]$  addieren und  $\text{E}[X]^2$  von der rechten Seite subtrahieren, dann erhalten wir genau die Varianz  $\text{Var}[X]$ .  $\square$

Wir betrachten als Nächstes einige wichtige Verteilungen.

**Uniforme Verteilung:** In der uniformen Verteilung sind alle Stichproben gleichwahrscheinlich und wir definieren dementsprechend diese „einfachste“ Verteilung durch

$$p_k = \text{prob}[X = k] = \frac{1}{|\Omega|}.$$

Wir sagen, dass  $X$  *uniform verteilt* ist.

---

**Aufgabe 4**

Berechne den Erwartungswert und die Varianz einer uniform verteilten Zufallsvariable.

---

**Binomialverteilung:** Wir sagen, dass eine Zufallsvariable  $X$  mit Parametern  $n$  and  $k$  *binomiell verteilt* ist, vorausgesetzt

$$\text{prob}[X = k] = \binom{n}{k} p^k (1-p)^{n-k}$$

gilt für jedes  $0 \leq k \leq n$ . Für  $n$  unabhängige Zufallsvariablen  $X_1, \dots, X_n$  mit  $\text{prob}[X_i = 1] = p$  und  $\text{prob}[X_i = 0] = 1 - p$  folgt  $\text{prob}[X_1 + \dots + X_n = k] = \binom{n}{k} p^k (1-p)^{n-k}$  und die Zufallsvariable  $X = X_1 + \dots + X_n$  ist binomiell verteilt.

Der Erwartungswert von  $X$  ist  $E[X] = \sum_{i=1}^n E[X_i] = np$  und auch die Varianz kann einfach berechnet werden. Wegen der Linearität des Erwartungswerts ist

$$E[X^2] = E\left[\left(\sum X_i\right)^2\right] = \sum_i E[X_i^2] + \sum_{i \neq j} E[X_i \cdot X_j].$$

Da  $X_i, X_j$  paarweise unabhängig sind, folgt  $E[X_i \cdot X_j] = E[X_i] \cdot E[X_j]$  und deshalb ist

$$E[X^2] = np + 2 \cdot \binom{n}{2} \cdot p^2 = np + n(n-1)p^2 = (np)^2 + np(1-p).$$

Also gilt  $\text{Var}[X] = E[X^2] - (E[X])^2 = np(1-p)$ .

**Geometrische Verteilung:** Wir werfen solange eine Münze, bis wir zum ersten Mal „Wappen“ erhalten. Wenn  $X$  die Anzahl der Versuche misst, dann ist  $X$  auf der Menge  $\Omega = \{\text{Zahl}^n \cdot \text{Wappen} \mid n \in \mathbb{N}\}$  definiert und wir haben zum ersten Mal einen Stichprobenraum von abzählbar unendlicher Größe erhalten. Wenn  $p$  die Wahrscheinlichkeit für „Wappen“ ist, dann erhalten wir

$$p_k = \text{prob}[X = k] = q^{k-1} \cdot p \quad \text{mit } q = 1 - p.$$

Wir benutzen Erzeugendenfunktionen, um Erwartungswert und Varianz zu berechnen. Die Erzeugendenfunktion von  $X$  ist

$$F_X(x) = \sum_{t=1}^{\infty} q^{t-1} p \cdot x^t = px \cdot \sum_{t=0}^{\infty} q^t x^t = \frac{px}{1-qx}.$$

Die erste und zweite Ableitung ist

$$F'_X(x) = \frac{(1-qx)p + pxq}{(1-qx)^2} = \frac{p}{(1-qx)^2}$$

und

$$F''_X(x) = \frac{2pq}{(1-qx)^3}.$$

Also folgt nach Satz 1.7,

$$E[X] = F'_X(1) = \frac{p}{(1-q)^2} = \frac{1}{p}$$

und

$$\text{Var}[X] = F_X''(1) + E[T] - E[Y]^2 = \frac{2pq}{p^3} + \frac{1}{p} - \frac{1}{p^2} = \frac{1-p}{p^2}.$$

#### Aufgabe 5

Eine *Zufallsquelle*  $Q$  produziert Nullen und Einsen, wobei eine Eins mit Wahrscheinlichkeit  $p$  erscheint. Wir können auf  $Q$  zugreifen, kennen aber  $p$  nicht. Unsere Aufgabe ist die Konstruktion einer perfekten Zufallsquelle, die eine Eins mit Wahrscheinlichkeit  $\frac{1}{2}$  produziert. Wir können also den Strom der Zufallsbits von  $Q$  beobachten und müssen einen Strom von perfekten Zufallsbits konstruieren.

Entwirf eine perfekte Zufallsquelle, so dass die erwartete Anzahl von  $Q$ -Bits pro perfektem Zufallsbit höchstens  $\frac{1}{p \cdot (1-p)}$  beträgt.

**Hinweis:** Betrachte zwei aufeinanderfolgende Bits von  $Q$ .

#### Aufgabe 6

Wir können auf eine perfekte Zufallsquelle zugreifen, die 0 und 1 mit Wahrscheinlichkeit genau  $\frac{1}{2}$  produziert.

- Wir möchten die Verteilung  $(p_1, \dots, p_n)$  erzeugen. Entwirf einen Algorithmus, der  $i$  mit Wahrscheinlichkeit  $p_i$  erzeugt und benutze höchstens die erwartete Anzahl von  $O(\log_2 n)$  Bits der perfekten Quelle. Beachte, dass  $p_1, \dots, p_n \geq 0$  beliebige reelle Zahlen sind, die zu Eins summieren.
- Bestimme die Worst-Case Anzahl perfekter Zufallsbits für  $p_1 = p_2 = p_3 = \frac{1}{3}$ .

#### Aufgabe 7

Wir spielen gegen einen Gegner, der sich zwei Zahlen ausdenkt und jede Zahl, für uns nicht sichtbar, auf jeweils ein eigenes Blatt Papier schreibt. Wir wählen zufällig ein Blatt, lesen die Zahl und haben dann die Wahl, die Zahl zu behalten oder gegen die verbleibende Zahl zu tauschen. Sei  $x$  die Zahl für die wir uns letztlich entscheiden und sei  $y$  die verbleibende Zahl. Dann ist  $x - y$  unser (möglicherweise negativer) Gewinn.

- Wir betrachten die Strategie  $S_t \equiv$  „Gib Zahlen  $< t$  zurück und behalte Zahlen  $\geq t$ “. Analysiere den erwarteten Gewinn  $E_{x,y}(\text{Gewinn}(S_t))$  dieser Strategie in Abhängigkeit von  $t, x$  und  $y$ .
- Entwirf eine randomisierte Strategie mit erwartetem positiven Gewinn für beliebige, aber verschiedene Zahlen  $x \neq y$ .

#### Aufgabe 8

Wir spielen ein Spiel mit Erfolgswahrscheinlichkeit  $p = 1/2$ . Wenn wir gewinnen, verdoppeln wir unseren Einsatz, wenn wir verlieren, verlieren wir auch unseren Einsatz. Betrachte die folgende Strategie

`i:=0`

`REPEAT`

`Setze den Einsatz  $2^i$  $`

`i:=i+1`

`UNTIL(Ich gewinne zum ersten Mal)`

Berechne den erwarteten Gewinn und den erwarteten Geldbetrag, um diese Strategie durchzuführen.

#### Beispiel 1.3 Berechnung des durchschnittlichen Gehalts ohne Offenlegung der Einzelgehälter.

$n$  Personen  $1, \dots, n$  möchten das durchschnittliche Gehalt bestimmen, ohne aber ihr eigenes Gehalt offen legen zu müssen. Wenn alle Personen ehrlich sind, dann sollte das durchschnittliche Gehalt auch korrekt berechnet werden. Wenn jedoch irgendwelche  $k$  Personen lügen, dann sollten Sie keinerlei Information bis auf die Summe aller Gehälter gewinnen.

#### Algorithmus 1.8 Ein sicheres Protokoll

- (1)  $M$  sei eine genügend große Zahl, die größer als die Summe aller Gehälter ist.

Jede Person  $i$  zufällig wählt Zahlen  $X_{i,1}, \dots, X_{i,i-1}, X_{i,i+1}, \dots, X_{i,n} \in \{0, \dots, M-1\}$  und teilt  $X_{i,j}$  der Person  $j$  mit.

(2) Wenn  $G_i$  das Gehalt von Person  $i$  ist, dann bestimmt sie die Restklasse

$$S_i = G_i + \sum_{j,j \neq i}^n X_{j,i} - \sum_{j,j \neq i}^n X_{i,j} \pmod{M}.$$

(3) Jede Person  $i$  veröffentlicht  $S_i$  und  $\sum_i S_i \pmod{M}$  wird berechnet.

*Kommentar:* Wenn alle Personen ehrlich sind, dann ist

$$\sum_i S_i \pmod{M} \equiv \sum_i G_i + \sum_{i,j,j \neq i}^n X_{j,i} - \sum_{i,j,j \neq i}^n X_{i,j} \pmod{M} \equiv \sum_i G_i \pmod{M} = \sum_i G_i$$

und  $\frac{1}{n} \cdot \sum_j S_j$  ist das gewünschte durchschnittliche Gehalt.

Warum ist dieses Protokoll sicher? Angenommen, die letzten  $k$  Personen sind Lügner. Wir setzen  $S_i^* = G_i + \sum_{j=1, j \neq i}^{n-k} X_{j,i} - \sum_{j=1, j \neq i}^{n-k} X_{i,j} \pmod{M}$ . Eine ehrliche Person veröffentlicht

$$S_i = S_i^* + \sum_{j=n-k+1}^n X_{j,i} - \sum_{j=n-k+1}^n X_{i,j} \pmod{M}$$

Beachte, dass  $\sum_{i=1}^{n-k} S_i^* = \sum_{i=1}^{n-k} G_i$  gilt.

---

#### Aufgabe 9

Zeige: Jede Kombination  $(s_2^*, \dots, s_{n-k}^*)$  von Werten der Zufallsvariable  $S_2^*, \dots, S_{n-k}^*$  wird mit Wahrscheinlichkeit  $M^{-(n-k-1)}$  erzeugt. Deshalb sind die Zufallsvariablen  $S_2^*, \dots, S_{n-k}^*$  unabhängig.

---

Nach dieser Aufgabe sind die Zufallsvariablen  $S_2^*, \dots, S_{n-k}^*$  unabhängig und uniform über der Menge  $\{0, \dots, M-1\}$  verteilt. Diese Aussage gilt aber mit dem selben Argument auch für  $S_2, \dots, S_{n-k}$  und die Lügner lernen nichts, da jede Folge von  $n-k-1$  Werten mit Wahrscheinlichkeit  $\frac{1}{M^{n-k-1}}$  auftritt.

Abweichungen vom Erwartungswert kann man mit Hilfe der Ungleichungen von Markoff, Chernoff and Tschebyscheff abschätzen. Wir beginnen mit Markoff's Ungleichung.

**Markoff's Ungleichung:** Sei  $X$  eine nicht-negative Zufallsvariable und sei  $a > 0$  eine reelle Zahl. Dann gilt

$$\text{prob}[X > a] \leq \frac{\text{E}[X]}{a}.$$

**Beweis:** Nach Definition des Erwartungswerts ist

$$\text{E}[X] = \sum_x x \cdot \text{prob}[X = x] \geq \sum_{x>a} a \cdot \text{prob}[X = x] = a \cdot \text{prob}[X > a].$$

□

**Tschebyscheff's Ungleichung:**  $X$  sei eine Zufallsvariable. Dann gilt

$$\text{prob}[|X - \text{E}[X]| > t] \leq \frac{\text{Var}[X]}{t^2}.$$

für jede reelle Zahl  $t > 0$ .

**Beweis:** Wir definieren die Zufallsvariable  $Y = (X - E[X])^2$  und wenden Markoff's Ungleichung an. Das ergibt

$$\text{prob}[|X - E[X]| > t] = \text{prob}[Y > t^2] \leq E[Y]/t^2 = \text{Var}[X]/t^2.$$

□

**Chernoff's Ungleichungen** sind Spezialfälle der Markoff Ungleichung, angewandt auf Summen von unabhängigen 0-1 Zufallsvariablen.

**Satz 1.9**  $X_1, \dots, X_n$  seien beliebige unabhängige binäre Zufallsvariablen mit den Erfolgswahrscheinlichkeit  $p_i = \text{prob}[X_i = 1]$ . Dann ist  $N = \sum_{i=1}^n p_i$  die erwartete Anzahl der Erfolge und es gilt

$$\begin{aligned} \text{prob}\left[\sum_{i=1}^n X_i > (1 + \beta) \cdot N\right] &\leq \left(\frac{e^\beta}{(1 + \beta)^{1+\beta}}\right)^N \leq e^{-N \cdot \beta^2/3} \\ \text{prob}\left[\sum_{i=1}^n X_i < (1 - \beta) \cdot N\right] &\leq \left(\frac{e^{-\beta}}{(1 - \beta)^{1-\beta}}\right)^N \leq e^{-N \cdot \beta^2/2} \end{aligned}$$

für jedes  $\beta > 0$  (bzw.  $0 < \beta \leq 1$  im zweiten Fall).

**Beweis:** Wir zeigen nur die erste Ungleichung. Wir wenden die Markoff Ungleichung für beliebiges  $\alpha > 0$  an und erhalten

$$\begin{aligned} \text{prob}\left[\sum_{i=1}^n X_i > t\right] &= \text{prob}\left[e^{\alpha \cdot \sum_{i=1}^n X_i} > e^{\alpha \cdot t}\right] \\ &\leq e^{-\alpha \cdot t} \cdot E\left[e^{\alpha \cdot \sum_{i=1}^n X_i}\right] \\ &= e^{-\alpha \cdot t} \cdot \prod_{i=1}^n E\left[e^{\alpha \cdot X_i}\right]. \end{aligned}$$

In der letzten Gleichung haben wir benutzt, dass  $E[Y_1 \cdot Y_2] = E[Y_1] \cdot E[Y_2]$  gilt, wenn  $Y_1$  und  $Y_2$  unabhängige Zufallsvariablen sind. Wir ersetzen  $t$  durch  $(1 + \beta) \cdot N$ ,  $\alpha$  durch  $\ln(1 + \beta)$  und erhalten

$$\begin{aligned} \text{prob}\left[\sum_{i=1}^n X_i > (1 + \beta) \cdot N\right] &\leq e^{-\ln(1+\beta) \cdot N} \cdot \prod_{i=1}^n E\left[e^{\ln(1+\beta) \cdot X_i}\right] \\ &= (1 + \beta)^{-(1+\beta) \cdot N} \cdot \prod_{i=1}^n E\left[(1 + \beta)^{X_i}\right]. \end{aligned}$$

Die Behauptung folgt, da  $E\left[(1 + \beta)^{X_i}\right] = p_i(1 + \beta) + (1 - p_i) = 1 + \beta \cdot p_i \leq e^{\beta \cdot p_i}$  gilt. □

---

#### Aufgabe 10

Sei  $x \leq 1$  eine beliebige reelle Zahl. Zeige

$$\frac{e^x}{(1+x)^{1+x}} = e^{x - (1+x)\ln(1+x)} \leq e^{-x^2/3}.$$


---

**Aufgabe 11**

$X_1, \dots, X_n$  seien unabhängige, binäre Zufallsvariablen mit  $p_i = \text{prob}[X_i = 1]$  und nimm an, dass  $N^* \geq \sum_{i=1}^n p_i$  gilt. Zeige, dass

$$\text{prob} \left[ \sum_{i=1}^n X_i > (1 + \beta) \cdot N^* \right] \leq e^{-\beta^2 \cdot N^* / 3}$$

für jedes  $\beta > 0$  folgt.

**Aufgabe 12**

Zeige die zweite Chernoff-Ungleichung.

**Beispiel 1.4** Wir nehmen an, dass ein Zufallsexperiment  $X$  vorliegt, dessen Erwartungswert wir experimentell messen möchten. Das Experiment sei aber instabil, d.h. die Varianz von  $X$  ist groß.

Wir „boosten“, wiederholen also das Experiment  $k$  mal. Wenn  $X_i$  das Ergebnis des  $i$ ten Experiments ist, dann setzen wir  $Y = \frac{1}{k} \cdot \sum_{i=1}^k X_i$  und beachten, dass die Zufallsvariablen  $X_1, \dots, X_k$  unabhängig sind: Es gilt also

$$V[Y] = \frac{1}{k^2} \cdot \sum_{i=1}^k V[X_i] = \frac{1}{k^2} \cdot \sum_{i=1}^k V[X] = \frac{1}{k} \cdot V[X].$$

Wir haben die Varianz um den Faktor  $k$  gesenkt, aber den Erwartungswert unverändert gelassen, denn  $E[Y] = E[\frac{1}{k} \cdot \sum_{i=1}^k X_i] = \frac{1}{k} \cdot \sum_{i=1}^k E[X] = E[X]$ . Die Tschebyscheff Ungleichung liefert jetzt das Ergebnis

$$\text{prob}[|Y - E[X]| > t] = \text{prob}[|Y - E[Y]| > t] \leq \frac{V[Y]}{t^2} = \frac{V[X]}{k \cdot t^2}$$

und große Abweichungen vom Erwartungswert sind unwahrscheinlicher geworden.

Angenommen, wir haben erreicht, dass  $Y$  mit Wahrscheinlichkeit mindestens  $p = 1 - \varepsilon$  in ein „Toleranzintervall“  $T = [E[X] - \delta, E[X] + \delta]$  fällt. Können wir  $p$  „schnell gegen 1 treiben“? Wir wiederholen diesmal das Experiment  $Y$  und zwar  $m$  mal und erhalten wiederum unabhängige Zufallsvariablen  $Y_1, \dots, Y_m$ . Als Schätzung des Erwartungswerts geben wir jetzt den *Median*  $M$  von  $Y_1, \dots, Y_m$  aus. Wie groß ist die Wahrscheinlichkeit, dass  $M$  nicht im Toleranzintervall  $T$  liegt?

$Y$  liegt mit Wahrscheinlichkeit mindestens  $p$  in  $T$ . Wenn also der Median außerhalb des Toleranzintervalls liegt, dann liegen mindestens  $\frac{m}{2}$  Einzelschätzungen außerhalb, während nur  $(1 - p) \cdot m = \varepsilon \cdot m$  außerhalb liegende Einzelschätzungen zu erwarten sind. Wir wenden die Chernoff Ungleichung an und beachten, dass  $(1 + \frac{1-2\varepsilon}{2\varepsilon}) \cdot \varepsilon \cdot m = \frac{m}{2}$ . Also erhalten wir mit  $\beta = \frac{1-2\varepsilon}{2\varepsilon}$

$$\text{prob}[M \notin T] \leq e^{-\varepsilon \cdot m \cdot \beta^2 / 3} = e^{-(1-2\varepsilon)^2 \cdot m / (12\varepsilon)}$$

und die Fehlerwahrscheinlichkeit fällt negativ exponentiell, falls  $\varepsilon < \frac{1}{2}$ .

## 1.5 Lineare Programmierung

$x_1, \dots, x_n$  seien  $n$  reellwertige Variablen. In der linearen Programmierung ist ein System linearer Gleichungen

$$\sum_{j=1}^n a_{ij} x_j = b_i$$

und/oder linearer Ungleichungen

$$\sum_{j=1}^n a_{ij}x_j \geq b_i \quad \text{bzw.} \quad \sum_{j=1}^n a_{ij}x_j \leq b_i$$

zu erfüllen. Für eine jede Variable  $x_i$  ist zusätzlich zu entscheiden, ob die Nichtnegativitätsbedingung  $x_i \geq 0$  gefordert wird. Unter allen Lösungsvektoren  $x := (x_1, \dots, x_n)$  suchen wir dann einen Vektor, der die lineare Zielfunktion

$$c^T \cdot x = \sum_{j=1}^n c_j x_j$$

minimiert, bzw. maximiert. Wenn wir insgesamt  $m$  Gleichungen und/oder Ungleichungen festlegen, dann ist

$$A = [a_{ij}]_{1 \leq i \leq m, 1 \leq j \leq n}$$

die  $m \times n$ -Matrix des Systems von Gleichungen und Ungleichungen. Wir definieren  $b$  als den Vektor der rechten Seiten,  $c$  als den Koeffizientenvektor der linearen Zielfunktion und  $\square \in \{=, \geq\}^m$  als den Vektor der Vergleichsoperatoren. Das allgemeine Minimierungsproblem hat dann die Kurzform

$$\text{minimiere } c^T \cdot x, \quad \text{so dass } A \cdot x \square b \tag{1.7}$$

und für alle  $i$ :  $x_i \geq 0$  oder  $x_i$  ist unbeschränkt.

Dieses Minimierungsproblem kann mit dem Simplex-Algorithmus bzw. mit Interior-Point Verfahren gelöst werden, wobei Interior-Point Verfahren eine garantiert polynomielle Laufzeit besitzen. (Die verbrauchte Zeit wird in Abhängigkeit von der Länge der Binärdarstellung von  $A, b$  und  $c$  gemessen). Dem Minimierungsproblem (1.7) ist ein duales Maximierungsproblem zugeordnet:

$$\text{maximiere } b^T \cdot y, \quad \text{so dass } y^T \cdot A \square^* c^T \tag{1.8}$$

und für alle  $j$ :  $y_j \geq 0$  oder  $y_j$  ist unbeschränkt.

Die Wahl der Vergleichsoperatoren wie auch die Wahl der Nichtnegativitätsbedingungen ist abhängig vom „primalen“ Programm (1.7). Um diese Abhängigkeit zu erklären, ordnen wir die primale Variable  $x_i$  der  $i$ ten Bedingung des dualen Programms und die duale Variable  $y_j$  der  $j$ ten Bedingung des primalen Programms zu.

- Wenn die  $j$ te Bedingung des primalen Programms eine Gleichung ist ( $\square_j$  stimmt mit  $=$  überein), dann ist  $y_j$  uneingeschränkt. Wenn die  $j$ te Bedingung eine Ungleichung ist ( $\square_j$  stimmt mit  $\geq$  überein), dann ist  $y_j \geq 0$  zu fordern.
- Wenn  $x_i$  unbeschränkt ist, dann ist die  $i$ te duale Bedingung eine Gleichung. Wenn hingegen  $x_i \geq 0$  gefordert wird, dann ist eine Ungleichung zu wählen, wobei  $\square_j^*$  mit  $\leq$  übereinstimmen muss.

Das primale und das duale Programm haben überraschenderweise identische Optima!

**Satz 1.10 Dualitäts-Theorem**

Sei  $x$  eine Lösung des primalen Problems (1.7) und  $y$  eine Lösung des dualen Problems (1.8), sowie  $x_{\text{opt}}$  und  $y_{\text{opt}}$  optimale Lösungen von (1.7) bzw. (1.8). Dann gilt

(a) schwache Dualität:  $c^T \cdot x \geq b^T \cdot y$  und

(b) starke Dualität:  $c^T \cdot x_{\text{opt}} = b^T \cdot y_{\text{opt}}$ .

Wir zeigen in Satz ??, dass das von Neumann'sche Minimax-Theorem eine Konsequenz des Dualitätstheorems ist. Eine weitere fundamentale Konsequenz ist die komplementäre Slackness. Dazu betrachten wir das primale Programm

$$(P) \text{ minimiere } c^T \cdot x, \text{ so dass } A \cdot x = b \text{ und } x \geq 0.$$

Dann ist

$$(D) \text{ maximiere } y^T \cdot b, \text{ so dass } y^T \cdot A \leq c^T$$

das duale Programm und beide Optima, wenn endlich, stimmen überein. Wir modifizieren das duale Programm (D) durch die Einführung von „Slack“-Variablen  $s$  und erhalten

$$(D) \text{ maximiere } y^T \cdot b, \text{ so dass } y^T \cdot A + s^T = c^T \text{ und } s \geq 0.$$

Die *Dualitätslücke*  $c^T \cdot x - b^T \cdot y$  gibt den Abstand zwischen der Lösung  $x$  des primalen und der Lösung  $(y, s)$  des dualen Problems an. Es ist

$$\begin{aligned} c^T \cdot x - b^T \cdot y &= c^T \cdot x - (Ax)^T \cdot y \\ &= x^T \cdot c - x^T \cdot A^T \cdot y \\ &= x^T \cdot (c - A^T \cdot y) \\ &= x^T \cdot s. \end{aligned}$$

Wir erhalten deshalb als Konsequenz des Dualitäts-Theorems:

**Satz 1.11 Komplementäre Slackness**

Sei  $x$  eine Lösung des primalen Problems (P) und  $(y, s)$  eine Lösung des dualen Problems (D). Dann sind äquivalent

(a) Die Lösung  $x$  ist optimal für (P) und  $(y, s)$  ist optimal für (D).

(b)  $x^T \cdot s = 0$ .

(c) Für alle  $i$  gilt  $x_i \cdot s_i = 0$ .

(d) Aus  $s_i > 0$  folgt  $x_i = 0$ .

Für optimale Lösungen  $x$  und  $y$  des primalen, bzw. dualen Programms gilt also: Wenn eine duale Ungleichung „Slack“ hat (also wenn  $s_i > 0$ ), dann ist die entsprechende primale Variable des Ausgangsproblems gleich 0. Auch die umgekehrte Beziehung für die primalen Ungleichungen  $x \geq 0$  gilt: Wenn  $x_i > 0$ , dann hat die  $i$ te duale Ungleichung keinen Slack.



Teil I

# Die Bearbeitung großer Datenmengen



## Kapitel 2

# Suchmaschinen

Suchmaschinen müssen für eine Anfrage bestehend aus mehreren Stichworten die informativsten Dokumente für diese Stichworte bestimmen. Wir beschreiben zuerst die Architektur einer Suchmaschine und stellen dann den Random Surfer Ansatz von Google sowie den Hubs-Authorities Ansatz von Kleinberg vor. Diese beiden Ansätze sind Vertreter der *Connectivity-Analyse*, die sich auf die beiden folgenden Annahmen gründet: Wenn ein Dokument  $A$  auf Dokument  $B$  zeigt, dann

- gibt es eine inhaltliche Beziehung zwischen den beiden Dokumenten und
- der Autor des Dokuments  $A$  hält Dokument  $B$  für wertvoll.

Beide Ansätze versuchen, die relative Wertschätzung zwischen Dokumenten in eine absolute Relevanz der Dokumente umzurechnen und damit ergibt sich die Möglichkeit, Dokumente aufgrund der Graphstruktur des WWW zu bewerten. Wir schliessen mit einer allgemeinen Behandlung von Methoden für den Entwurf von Meta-Suchmaschinen.

### 2.1 Die Architektur von Suchmaschinen

Google hat mittlerweile ungefähr 8 Milliarden Webseiten (oder Dokumente) indiziert. Da die durchschnittliche Größe eines Dokuments zwischen 5 und 10 Kilobytes liegt, bewegt sich damit die Gesamtgröße des indizierten Webs auf die 100 Terabyte zu. Desweiteren wird geschätzt, dass im .com Bereich 40% aller Dokumente täglich geändert werden; rund die Hälfte aller Webseiten scheint eine Lebenszeit von nur 10 Tagen zu besitzen.

Diese Zahlen unterstreichen die Komplexität des Suchproblems: Für einen sich rasant ändernden Suchraum gigantischer Größe sind Anfragen ohne merkliche Reaktionszeit zu beantworten. Suchmaschinen stellen sich diesen Anforderungen mit Hilfe der folgenden Komponenten:

- Der *Crawler* durchforstet das Web, um neue oder veränderte Dokumente zu bestimmen.
- *Abspeichern der wichtigsten Informationen*: Die vom Crawler gefundene Information muss aufbereitet und abgespeichert werden.
- *Indizierung*: Die Datenstruktur zum Abruf der abgespeicherten Information muss in Echtzeit alle Dokumente bestimmen, die die Anfrage enthalten.

- *Bewertung der Dokumente:* Die ausgewählten Dokumente sind im Hinblick auf ihren Informationsgehalt zu bewerten.

Der **Crawler** muss den Webgraphen durch eine Relevanz-gewichtete, parallel ausgeführte Breitensuche traversieren. Dabei sind verschiedenste Probleme lösen. Da eine Erfassung aller Webseiten illusorisch ist, ist zu entscheiden, welche Dokumente abzuspeichern sind. Welche Dokumente sollten in welchen Zeiträumen auf Veränderung untersucht werden? Auch hier sind die wichtigen, und unter diesen die in der Vergangenheit häufig veränderten Dokumente vorrangig zu behandeln. Wie sollte der Crawler die Belastung der besuchten Webseiten minimieren? Wie kann der Crawling Prozess optimal parallelisiert werden?

**Abspeichern der wichtigsten Informationen.** Das Web Repository wird vom Crawler aufgebaut und muss den direkten Zugriff auf Dokumente sowie das Einfügen und das Entfernen von Dokumenten unterstützen. Desweiteren müssen alle Dokumente mit einer gegebenen Eigenschaft, wie dem Besitz einer Menge von Stichworten, schnell bestimmbar sein. Diese Aufgaben werden oft mit einer Kombination von Hashing und B-Bäumen gelöst.

In der **Indizierung** werden alle Worte eines jeden Dokuments erfasst. Häufig wird auch ein Wörterbuch berechnet, das alle je erfassten Worte aufführt und statistische Informationen wie Häufigkeiten angibt. Eine fundamentale Datenstruktur in der Indizierung ist der *invertierte Index*, in dem zu jedem Stichwort  $s$  alle Dokumente bestimmt werden, die  $s$  enthalten. Üblicherweise enthält der invertierte Index noch Zusatzinformation, um die Prominenz des Stichworts innerhalb des Dokuments zu beschreiben: hier spielen die Häufigkeit des Stichworts, seine Schriftgröße wie auch das Vorkommen des Stichworts in Beschriftungen von Links auf das Dokument wichtige Rollen.

Neben diesen Text-basierten Indizes wird in modernen Suchmaschinen auch ein Link-Index berechnet, um mit der Graphstruktur des Webgraphen arbeiten zu können. Der Link-Index wird üblicherweise als Adjazenzliste abgespeichert.

Die **Bewertung der Dokumente** ist ein zentrales Hilfsmittel, um informativste Dokumente für eine gegebene Anfrage bestimmen zu können. Moderne Suchmaschinen basieren eine solche Bewertung auf Methoden des Peer-Review und sind somit nur relativ schwach von solchen Eigenschaften des Dokuments abhängig, die vom Autor des Dokuments direkt beeinflussbar sind. Wir beschreiben erfolgreiche Methoden des Peer-Review in den Abschnitten 2.2 und 2.3.

## 2.2 Google

Google bestimmt zuerst alle Dokumente, die die Stichworte der Anfrage enthalten. Die gefundenen Dokumente werden dann sowohl lokal wie auch global bewertet. In der lokalen oder anfrage-abhängigen Bewertung spielen Aspekte wie etwa

- Schriftgröße und Nähe der Stichworte zueinander innerhalb des Dokuments,
- Häufigkeit des Vorkommens der Stichworte innerhalb des Dokuments und
- Vorkommen der Stichworte in der Beschriftung von Hyperlinks, die auf das Dokument zeigen,

eine Rolle.

In der globalen oder anfrage-unabhängigen Bewertung wird der *Page-Rank* des Dokuments ermittelt. Der Page-Rank  $\text{pr}(w)$  eines Dokuments  $w$  soll die Qualität des Dokuments wiedergeben und wird im Wesentlichen durch einen „Peer-Review“ bestimmt: Der Page-Rank  $\text{pr}(w)$  ist hoch, wenn viele Dokumente  $u$  mit hohem Page-Rank  $\text{pr}(u)$  auf das Dokument  $w$  zeigen. Ein erster Ansatz zur Bestimmung des Page-Ranks nimmt an, dass ein Dokument  $u$  mit  $d_u$  Hyperlinks seinen Page-Rank gleichmäßig verteilt: Wenn  $u$  auf Dokument  $w$  zeigt, dann „erbt“  $w$  den Bruchteil  $\frac{\text{pr}(u)}{d_u}$  und  $w$  erbt insgesamt den Betrag

$$\text{pr}(w) = \sum_{u \text{ zeigt auf } w} \frac{\text{pr}(u)}{d_u} \quad (2.1)$$

Um diese Definition zu interpretieren, fassen wir das WWW als eine Markov-Kette auf.

**Definition 2.1** Eine Markoff-Kette  $(G, P)$  besteht aus einem gerichteten Graphen  $G = (V, E)$  und einer Übergangsmatrix  $P$ , deren Zeilen und Spalten mit Knoten aus  $V$  indiziert sind. Für jeden Knoten  $v \in V$  gilt

$$\sum_{w \in V} P[v, w] = 1.$$

Zusätzlich ist  $P[v, w] = 0$  für  $(v, w) \notin E$ . (Eine Matrix  $P$  mit  $P \geq 0$  und  $\sum_j P[i, j] = 1$  für alle  $i$  heißt *stochastische Matrix*.)

Die Knoten der „WWW-Kette“ entsprechen den Dokumenten und eine Kante  $(u, w)$  entspricht einem Hyperlink von Dokument  $u$  auf Dokument  $w$ . Wir nehmen an, dass ein „Random-Surfer“ einen Hyperlink nach der Gleichverteilung auswählt und demgemäß wird ein Nachbar des Dokuments  $u$  mit Wahrscheinlichkeit  $\frac{1}{d_u}$  gewählt. Dabei unterstellt Google, dass jedes Dokument einen Hyperlink auf sich selbst hat, so dass  $d_u \geq 1$  für alle Dokumente  $u$  gilt. Wir erhalten somit die Übergangsmatrix  $P$  mit

$$P[u, w] = \begin{cases} \frac{1}{d_u} & (u, w) \text{ ist ein Hyperlink} \\ 0 & \text{sonst.} \end{cases}$$

Die Page-Rank Definition (2.1) ist jetzt äquivalent zu

$$\text{pr} = \text{pr} \cdot P. \quad (2.2)$$

Diese Definition erinnert stark an die Definition einer stationären Verteilung:

**Definition 2.2** Sei  $(G, P)$  eine Markoff-Kette mit  $G = (V, E)$ . Dann heißt eine Verteilung  $\pi = (\pi_v \mid v \in V)$  auf  $V$  eine stationäre Verteilung, falls  $\pi = \pi \cdot P$ .

Der Page-Rank entspricht somit einer stationären Verteilung der WWW-Kette. Warum sind stationäre Verteilungen wichtig? Wenn eine Markoff-Kette *ergodisch* ist, dann beschreibt die stationäre Verteilung die Grenzverteilung, also die Wahrscheinlichkeit mit der der Random-Surfer ein Dokument besucht, wenn ein beliebig langer Zeitraum zur Verfügung steht.

**Definition 2.3** Die Markoff-Kette  $(G, P)$  heißt genau dann ergodisch, wenn für alle  $i_1, i_2$  und  $j$  die Grenzwerte  $\lim_{k \rightarrow \infty} P^k[i_1, j]$  und  $\lim_{k \rightarrow \infty} P^k[i_2, j]$  existieren und

$$\lim_{k \rightarrow \infty} P^k[i_1, j] = \lim_{k \rightarrow \infty} P^k[i_2, j] > 0$$

gilt.

Für eine ergodische Markoff-Kette ist die Matrix

$$P^\infty = \lim_{k \rightarrow \infty} P^k = \left( \lim_{k \rightarrow \infty} P^k[i, j] \right)_{i, j \in V}$$

wohldefiniert, da wir voraussetzen, dass die Grenzwerte existieren. Weiterhin fordern wir, dass alle Zeilen in  $P^\infty$  identisch sind: Wenn wir einen Startknoten gemäß einer beliebigen Anfangsverteilung  $\pi^{(0)}$  wählen und hinreichend viele zufällige Schritte machen, ist die Wahrscheinlichkeit einen bestimmten Knoten  $j$  zu erreichen, stets beliebig nahe an  $P^\infty[1, j] = \dots = P^\infty[n, j]$ . Die Wahl des Anfangsknoten ist für einen hinreichend langen Random Walk also ohne Belang und die Grenzverteilung einer ergodischen Markoff-Kette „vergisst“ somit den Startpunkt eines Random Walks.

Wir definieren die Grenzverteilung  $\pi^{(\infty)}$  als die Verteilung der ersten Zeile.

**Fakt 2.1** *Jede ergodische Markoff-Kette  $(G, P)$  besitzt eine eindeutig bestimmte stationäre Verteilung  $\pi$  und es gilt  $\pi = \pi^{(\infty)}$ .*

Leider können wir nicht erwarten, dass die WWW-Kette ergodisch ist, denn der Random Surfer wird zum Beispiel von Dokumenten ohne Hyperlinks gefangen und die Grenzverteilung vergisst somit den Startpunkt *nicht*.

Um zu einer ergodischen Kette zu gelangen, verändert Google deshalb die Sichtweise ein wenig. Im erfolgreichen zweiten Ansatz legt Google einen Random Walk zugrunde, der mit Wahrscheinlichkeit  $(1 - d)$  ein benachbartes Dokument aufsucht und mit Wahrscheinlichkeit  $d$  zu einem zufälligen Dokument springt. Wenn wir annehmen, dass es genau  $n$  Dokumente gibt, dann werden wir auf die Übergangsmatrix

$$P'[u, w] = \begin{cases} \frac{d}{n} + \frac{1-d}{d_u} & (u, w) \text{ ist ein Hyperlink} \\ \frac{d}{n} & \text{sonst.} \end{cases}$$

geführt.

---

### Aufgabe 13

Zeige, dass die modifizierte WWW-Kette ergodisch ist.

Hinweis: Es genügt zu zeigen, dass die Kette irreduzibel und aperiodisch ist. (Die Kette ist irreduzibel, wenn jeder Knoten jeden anderen Knoten erreichen kann. Die Kette ist aperiodisch, wenn der größte gemeinsame Teiler aller Weglängen zwischen zwei beliebigen Knoten mit Eins übereinstimmt.)

---

Wir fordern wiederum, dass der Page-Rank der (diesmal eindeutigen) stationären Verteilung entspricht und erhalten deshalb wegen  $\text{pr} = \text{pr} \cdot P'$  und  $\sum_u \text{pr}(u) = 1$  die Setzung

$$\begin{aligned} \text{pr}(w) &= \sum_u \frac{d}{n} \cdot \text{pr}(u) + (1-d) \cdot \sum_{u \text{ zeigt auf } w} \frac{\text{pr}(u)}{d_u} \\ &= \frac{d}{n} + (1-d) \cdot \sum_{u \text{ zeigt auf } w} \frac{\text{pr}(u)}{d_u}. \end{aligned}$$

In Matrix-Vektor Notation erhalten wir deshalb

$$\text{pr} = d \cdot p + (1-d) \cdot \text{pr} \cdot P, \quad (2.3)$$

wobei der Vektor  $p$  durch  $p_w = \frac{1}{n}$  definiert ist

---

### Aufgabe 14

Ein Anbieter einer Internetseite  $w$  möchte seiner Webseite, die keine ausgehenden Hyperlinks hat, zu einem höheren Page-Rank verhelfen.

- (a) In einem ersten Versuch legt der Anbieter  $c$  neue Seiten an, die jeweils wechselseitig aufeinander und auf die ursprüngliche Seite  $w$  verweisen. Zeige, dass sich der Page-Rank seiner Seite  $w$  dadurch um höchstens  $(1 - d - \frac{1}{n}) \cdot \frac{cd}{n+c}$  erhöht.
- (b) In einem zweiten Versuch legt er  $c$  Seiten an, die alle ausschließlich auf die Seite  $w$  verweisen. Zeige, dass sich der Page-Rank seiner Seite  $w$  dadurch um höchstens  $(1 - d - \frac{1}{n}) \cdot \frac{cd}{n+c}$  erhöht.

Wie wird der Pagerank  $pr$  berechnet? Zuerst wird ausgenutzt, dass  $\lim_{k \rightarrow \infty} \pi_0^T \cdot (P^k) = pr$  für jede Anfangsverteilung  $\pi_0$  gilt. Wir wählen  $\pi_0$  als Gleichverteilung und setzen  $\pi_{k+1} := \pi_k^T \cdot P$ . Aufgrund des hohen Zusammenhangs des Netzes konvergiert  $\pi_k$  schnell gegen  $pr$ . Für eine schnelle Berechnung des Vektor-Matrix Produkts wird ausgenutzt, dass  $P$  viele Nulleinträge hat; desweiteren ist das Vektor-Matrix Produkt hochgradig parallelisierbar.

Gegenwärtig werden mehrere Tausend PC's eingesetzt, die mehrere Stunden zur Berechnung des Page-Ranks benötigen und selbst dieser Aufwand ist erstaunlich gering, da mehrere Milliarden Webseiten<sup>1</sup> involviert sind.

Da ein hoher Page-Rank finanziell sehr lukrativ ist, hat sich schon beinahe eine eigene Industrie zur Erhöhung des Page-Rank entwickelt. Diese Industrie firmiert unter dem Begriff der „Suchmaschinen-Optimierung“ (search engine optimization) und setzt neben legalen Methoden wie einer verbesserten inhaltlichen Darstellung auch illegale Spamming-Methoden ein: Für den Anwender nicht sichtbar<sup>2</sup>, werden beliebte Stichworte häufig wiederholt, um die lokale Bewertung der Webseite zu erhöhen. Desweiteren werden „Link Farms“<sup>3</sup> eingesetzt, um die globale Bewertung zu erhöhen. Obwohl Link Farms keinen absolut hohen Page-Rank erreichen können, so können jedoch stark eingeschränkte Suchanfragen „gewonnen“ werden. Google bestraft deshalb Seiten mit mehr als 100 ausgehenden Zeigern und senkt damit den vererbaren Page-Rank.

### 2.2.1 Inhalt-abhängiger Page-Rank

Es wird vermutet, dass Google mittlerweile einen „topic-sensitiven Page-Rank“ verwendet. Wir beschreiben hier einen Vorschlag aus [H02]. Die Grundidee ist die Berechnung von Page-Rank Vektoren  $pr^{(1)}, \dots, pr^{(K)}$  für jedes von  $K$  Themengebieten  $T_1, \dots, T_K$ :  $pr^{(i)}$  soll die Kompetenz von Dokumenten für das Themengebiet  $T_i$  messen.

(Zum Beispiel bieten sich die Themengebiete des Open Directory Projects an. Die allgemeinsten 16 Themengebiete sind hier *Arts* (Movies, Television, Music,...), *Business* (Jobs, Real Estate, Investing,...), *Computers* (Internet, Software, Hardware,...), *Games* (Video Games, RPGs, Gambling,...), *Health* (Fitness, Medicine,...), *Home* (Family, Consumers, Cooking,...), *Kids and Teens* (School, Teen Life,...), *News* (Media, Newspapers, Weather,...), *Recreation* (Travel, Food, Outdoors, Humor,...), *Reference* (Maps, Education, Libraries,...), *Regional* (US, Canada, UK, Europe,...), *Science* (Computer Science, Biology, Physics,...) *Shopping* (Autos, Clothing, Gifts,...), *Society* (People, Religion, Issues,...), *Sports* (Baseball, Soccer, Basketball,...), *World* (Deutsch, Español, Italiano, Polska, Dansk,...).

Der Page-Rank Vektor  $pr^{(i)}$  wird wiederum in einer **off-line Berechnung** bestimmt: Zuerst werden alle Dokumente im Hinblick auf das Themengebiet  $T_i$  bewertet; insbesondere sei  $p_w^{(i)}$  die Bewertung für Dokument  $w$ . Wir „personalieren“ die Page-Rank Berechnung (2.3), indem

<sup>1</sup>Anfang 2005 hat Google ungefähr 8 Milliarden Webseiten erfasst.

<sup>2</sup>Beliebt ist der Trick, Stichworte in der Hintergrundfarbe zu setzen. Alle guten Suchmaschinen kennen diese Tricks und disqualifizieren die betroffenen Webseiten.

<sup>3</sup>Jede Seite einer Link Farm zeigt auf viele andere Seiten der Farm.

wir  $p$  durch  $p^{(i)}$  ersetzen und erhalten

$$\text{pr}^{(i)} = d \cdot p^{(i)} + (1 - d) \cdot \text{pr}^{(i)} \cdot P, \quad (2.4)$$

als Grenzverteilung. Für die Bearbeitung der Suchanfrage  $Q = (Q_j | j)$  werden die bedingten Wahrscheinlichkeiten  $\text{prob}[Q_j | T_i]$  benötigt:  $\text{prob}[Q_j | T_i]$  wird **off-line** berechnet und misst die relevanz-gewichtete Häufigkeit mit der das Stichwort  $Q_j$  für das Themengebiet  $T_i$  auftritt. In der **on-line** Bearbeitung von  $Q$  werden dann die folgenden Schritte ausgeführt: Zuerst wird die Relevanz

$$\text{prob}[T_i | Q] = \frac{\Pi_j \text{prob}[Q_j | T_i] \cdot \text{prob}[T_i]}{\Pi_j \text{prob}[Q_j]}$$

des Themengebiets  $T_i$  für die Anfrage  $Q$  ermittelt. (Es ist nur  $\sum_i \text{prob}[T_i | Q] = 1$  zu fordern, deshalb ist eine Berechnung von  $\Pi_j \text{prob}[Q_j]$  nicht erforderlich.)

Danach werden alle Dokumente  $w$  ermittelt, in denen die Anfrage vorkommt. Wir bewerten dann  $w$  mit der Linearkombinationen aller Page-Ranks  $\text{pr}_w^{(i)}$ , gewichtet nach der jeweiligen Relevanz des Themengebiets. Wir setzen also

$$\text{Rang}(w | Q) = \sum_i \text{prob}[T_i | Q] \cdot \text{pr}_w^{(i)}.$$

## 2.3 Hubs und Authorities

Kleinberg [K1] stellt einen zweiten Ansatz für das Information Retrieval im WWW vor. Er unterscheidet *Hubs* (Dokumente mit „guten“ Links) und *Authorities* (Dokumente von hoher Qualität). Eine simultane Berechnung von Hubs und Authorities kann dann nach dem folgenden Muster erfolgen: Ein Dokument, das auf viele Dokumente mit hohem Authority-Gewicht zeigt, erhält ein hohes Hub-Gewicht, während ein Dokument, auf das viele Dokumente mit hohem Hub-Gewicht zeigen, ein hohes Authority-Gewicht erhält. Allerdings kennen wir anfänglich weder Hubs noch Authorities, und es sind somit Hubs *und* Authorities simultan zu bestimmen. Der HITS-Algorithmus (Hyperlink Induced Topic Search) von Kleinberg verarbeitet eine Anfrage  $\sigma$  wie folgt:

- (0)  $T$  sei ein fixierter Parameter.
- (1) Die Menge  $W_\sigma$  aller Dokumente, die die Stichworte der Anfrage enthalten, wird bestimmt. Aus diesen Dokumenten wird eine kleine Teilmenge  $R_\sigma$  relevanter Dokumente ausgewählt. (Kleinberg empfiehlt textbasierte Suchmaschinen wie Altavista oder Hotbot zur Durchführung der Auswahl. Eine textbasierte Suchmaschine bewertet ein Dokument unter Anderem nach der Häufigkeit und nach der Schriftgröße mit der die Stichworte der Anfrage erscheinen.)

In vielen Fällen werden die informationsreichsten Dokumente in der Menge  $R_\sigma$  nicht enthalten sein. Deshalb konstruiert der nächste Schritt eine größere Menge  $S_\sigma$ .

- (2) Setze  $S_\sigma = R_\sigma$ . Für jedes Dokument  $w \in R_\sigma$ :
  - (2a) Füge alle Dokumente zu  $S_\sigma$  hinzu, auf die  $w$  zeigt.
  - (2b) Füge alle Dokumente zu  $S_\sigma$  hinzu, die auf  $w$  zeigen. Wenn mehr als  $T$  Dokumente auf  $w$  zeigen, dann füge eine beliebige Auswahl von  $T$  Dokumenten hinzu. (Der Schwellenwert  $T$  ist geeignet zu bestimmen.)

- (3) Berechne Hub- und Authority-Gewichte.

Um Schritt (3) zu beschreiben, beschränken wir uns auf den Graphen  $G_\sigma = (S_\sigma, E_\sigma)$ , wobei die Kanten in  $E_\sigma$  genau den Hyperlinks zwischen Dokumenten aus  $S_\sigma$  entsprechen.

**Algorithmus 2.4 Berechnung der Hub- und Authority-Gewichte**

- (1) Es sei
- $n = |S_\sigma|$
- . Der Algorithmus berechnet zwei Vektoren
- $A$
- und
- $H$
- , wobei
- $A_u$
- (bzw.
- $H_u$
- ) die Qualität des Dokuments
- $u$
- als Authority (bzw. als Hub) misst.

Wir verlangen, dass  $A$  und  $H$  stets die Norm 1 besitzen und setzen anfänglich  $A = H = (1/\sqrt{n}, \dots, 1/\sqrt{n})^T$ .

- (2) Wiederhole genügend oft:

(2a) 
$$H_u = \sum_{w, (u,w) \in E_\sigma} A_w.$$

/\* Das Hub-Gewicht von  $u$  ist groß, wenn  $u$  auf viele Dokumente mit hohem Authority-Gewicht zeigt. \*/

(2b) 
$$A_u = \sum_{w, (w,u) \in E_\sigma} H_w.$$

/\* Das Authority-Gewicht von  $u$  ist groß, wenn viele Dokumente mit hohem Hub-Gewicht auf  $u$  zeigen. \*/

(2c) Normalisiere  $A$  und  $H$ , d.h. setze  $A = \frac{A}{\|A\|}$  und  $H = \frac{H}{\|H\|}$ .

Wir setzen  $x_0 = (1/\sqrt{n}, \dots, 1/\sqrt{n})^T$  und bezeichnen den Authority- bzw. den Hub-Vektor nach der  $i$ ten Iteration mit  $A^i$  bzw. mit  $H^i$ . Wenn  $M$  die Adjazenzmatrix von  $G_\sigma$  ist, dann erhalten wir ohne Normalisierung

$$H^{i+2} = M \cdot A^{i+1} \quad \text{und} \quad A^{i+1} = M^T \cdot H^i.$$

Also folgt

$$H^{i+2} = M \cdot A^{i+1} = M \cdot M^T \cdot H^i \quad \text{und} \quad A^{i+2} = M^T \cdot H^{i+1} = M^T \cdot M \cdot A^i$$

und deshalb ist

$$H^{2k} = (M \cdot M^T)^k \cdot x_0 \quad \text{und} \quad A^{2k} = (M^T \cdot M)^k \cdot x_0.$$

Beachte, dass  $M \cdot M^T$  wie auch  $M^T \cdot M$  symmetrische Matrizen sind. Symmetrische Matrizen besitzen reellwertige Eigenwerte wie auch eine Orthonormalbasis von Eigenvektoren:

**Fakt 2.2** Sei  $K$  eine symmetrische Matrix mit  $n$  Zeilen und Spalten.

- (a) Sämtliche Eigenwerte  $\lambda_1, \dots, \lambda_n$  von  $K$  sind reellwertig.
- (b) Es gibt eine Orthonormalbasis  $b_1, \dots, b_n$  von Eigenvektoren von  $K$ . (D.h. es ist  $K \cdot b_i = \lambda_i \cdot b_i$  und das innere Produkt  $\langle b_i, b_j \rangle$  verschwindet für  $i \neq j$  und ist Eins für  $i = j$ .)
- (c) Der betragsmäßig größte Eigenwert  $\lambda$  sei nichtnegativ und vom Betrag her größer als der betragsmäßig zweitgrößte Eigenwert. Wenn der Vektor  $x_0$  nicht senkrecht auf dem Eigenvektor  $v$  von  $\lambda$  steht, dann konvergiert die Folge  $(x_k \mid k)$  gegen  $\pm v$ , wobei

$$x_{k+1} = \frac{K \cdot x_k}{\|K \cdot x_k\|}$$

gelte.

Wir motivieren die Aussage (c). Nach Teil (b) gibt es eine Orthonormalbasis  $v_1, \dots, v_n$  aus Eigenvektoren zu den Eigenwerten  $\lambda_1 > \lambda_2 \geq \dots \geq \lambda_n$ . Wir können somit den Vektor  $x_0$  als Linearkombination

$$x_0 = \sum_{i=1}^n \alpha_i \cdot v_i$$

schreiben. Da der Basisvektor  $v_i$  Eigenvektor von  $K$  (zum Eigenwert  $\lambda_i$ ) ist, erhalten wir

$$K \cdot x_0 = \sum_{i=1}^n \alpha_i \cdot K \cdot v_i = \sum_{i=1}^n \alpha_i \cdot \lambda_i \cdot v_i$$

und nach  $t$ -facher Iteration

$$K^t \cdot x_0 = \sum_{i=1}^n \alpha_i \cdot \lambda_i^t \cdot v_i.$$

Da  $\lim_{t \rightarrow \infty} \lambda_i^t / \lambda_1^t = 0$  für  $i \neq 1$ , nimmt das Gewicht des Eigenvektors  $v_1$  am stärksten zu, und Aussage (c) folgt, wenn  $\alpha_1 \neq 0$  oder, alternativ, wenn gefordert wird, dass  $x_0$  nicht senkrecht auf  $v_1$  steht.

Als Konsequenz von Fakt 2.2 wird HITS unter sehr moderaten Annahmen gegen den größten Eigenvektor konvergieren, denn die symmetrischen und positiv semidefiniten Matrizen  $MM^T$  und  $M^T M$  haben nur nicht-negative Eigenwerte und der größte Eigenwert stimmt also mit dem betragsmäßig größten Eigenwert überein.

#### Aufgabe 15

Zeige: Wenn  $MM^T x = \lambda x$  gilt, dann ist  $\lambda \geq 0$ .

#### Aufgabe 16

Wir betrachten die symmetrische Matrix  $K = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ . Für welche Startvektoren  $x_0$  konvergiert die Folge  $(x_k | k \geq 0)$  mit  $x_{k+1} = \frac{K \cdot x_k}{\|K \cdot x_k\|}$ ?

**Korollar 2.5** Die größten Eigenwerte von  $M^T \cdot M$  und  $M \cdot M^T$  mögen die Vielfachheit 1 besitzen und  $x_0$  stehe nicht senkrecht auf den jeweils „größten“ Eigenvektoren. Dann bestimmt HITS Vektoren  $A$  bzw.  $H$ , die bis auf das Vorzeichen gegen die Eigenvektoren der größten Eigenwerte von  $M^T \cdot M$  bzw.  $M \cdot M^T$  konvergieren.

Wir schließen mit einem Vergleich zwischen Page-Rank und dem HITS Verfahren, bevor wir Schwachstellen des HITS Verfahrens im nächsten Abschnitt ansprechen.

- *Berechnungsaufwand*: Hier verbucht Page-Rank ein Plus, da es die Bewertungen vorbechnet. Dieser Vorteil ist jedoch nicht wesentlich, da von schneller Konvergenz (höchstens 20 Iterationen bei Mengen  $S_\sigma$  mit  $|S_\sigma| \approx 1000$ ) berichtet wird.
- *Spamming*: Auch hier scheint Page-Rank stärker, da HITS bei „verdorbener“ Auswahl  $R_\sigma$  verloren ist.
- *Qualität des Suchergebnisses*: Beide Verfahren liefern im Allgemeinen Ergebnisse guter Qualität. Da HITS die Suchanfrage in dem Connectivity-Ranking miteinbezieht, während Google Dokumente anfrage-unabhängig bewertet und die Suchanfrage nur in der lokalen Bewertung der Dokumente miteinbezieht, verdient HITS ein leichtes Plus.

### 2.3.1 Inhalt-abhängige Suche

In [BH] werden die folgenden Schwachstellen des HITS Verfahrens ausgemacht:

- (1) Angenommen, viele Dokumente  $A_i$  auf einem ersten Host zeigen auf ein einziges Dokument  $B$  auf einem zweiten Host. Als mögliche Konsequenz erhalten damit sowohl das Dokument  $B$  eine ungerechtfertigt hohe Authority-Gewichtung wie auch die Dokumente  $A_i$  eine ungerechtfertigt hohe Hub-Gewichtung.

Das gleiche Problem entsteht, wenn ein Dokument  $A$  auf dem ersten Host auf viele Dokumente  $B_i$  des zweiten Hosts zeigt.

- (2) Automatisch erzeugte Links verbinden Dokumente, die nicht notwendigerweise in einem thematischen Zusammenhang stehen und nicht-relevante Dokumente tauchen somit in der Menge  $S_\sigma$  auf. Jetzt besteht die Gefahr, dass nicht-relevante Dokumente dominieren: Zum Beispiel werden für die Suchanfrage „Jaguar AND Car“ Dokumente von Automobil-Herstellern ohne Bezug zur Marke Jaguar den größten Teil des Authority-Gewichts an sich reißen.

Die erste Schwachstelle kann durch eine einfache Modifikation von Hits beseitigt werden: Wenn  $z$  Dokumente  $A_i$  des ersten Hosts auf ein Dokument  $B$  des zweiten Hosts  $j$  zeigen, dann erhält jeder Zeiger das Authority-Gewicht  $a(A_i, B) = 1/z$ . Umgekehrt, wenn ein Dokument  $A$  des ersten Hosts auf  $z$  Dokumente  $B_1, \dots, B_z$  des zweiten Hosts zeigt, dann erhält jeder Zeiger das Hub-Gewicht  $h(A, B_j) = 1/z$ . Dementsprechend verändern wir die iterative Hub-Authority Berechnung wie folgt:

$$H_u = \sum_{w, (u,w) \in E_\sigma} h(u, w) \cdot A_w \quad \text{und} \quad A_u = \sum_{w, (w,u) \in E_\sigma} a(w, u) \cdot H_w.$$

---

#### Aufgabe 17

Zeige, dass die neue iterative Berechnung der Hub-Authority Gewichte konvergiert.

---

Um die zweite Schwachstelle abzustellen, wird versucht, das der Suchanfrage zugrunde liegende allgemeinere Themengebiet  $T$  zu bestimmen. In einem zweiten Schritt werden dann alle die Dokumente eliminiert, die für  $T$  nicht relevant sind. Das Themengebiet  $T$  wird nicht semantisch bestimmt, sondern  $T$  wird durch die Dokumente aus  $S_\sigma$ , definiert, die der Algorithmus als relevant für  $T$  ansieht.

Für die Relevanzmessung wird zuerst eine Auswahl von Dokumenten aus  $S_\sigma$  bestimmt. Die ersten 1000 Worte werden, über alle Dokumente der Auswahl, konkateniert und ergeben einen String  $T$ . Für jedes Dokument  $w \in S_\sigma$  wird dann die Ähnlichkeit zwischen  $T$  und  $w$  wie folgt bestimmt: Sowohl  $T$  wie auch  $w$  werden in Terme zerlegt und die Häufigkeiten  $h_{t,T}$  und  $h_{t,w}$  des Terms  $t$  in  $T$ , bzw.  $w$ , wird berechnet. Wenn  $h_t$  die Häufigkeit des Terms  $t$  über alle Dokumente des WWW ist, dann wird mit den Häufigkeitsvektoren

$$x_T = \left( \frac{h_{t,T}}{h_t} \mid t \right) \quad \text{und} \quad x_w = \left( \frac{h_{t,w}}{h_t} \mid t \right)$$

die Ähnlichkeit

$$\ddot{A}(T, w) = \frac{\langle x_T, x_w \rangle}{\|x_T\| \cdot \|x_w\|}$$

zwischen  $T$  und  $w$  definiert. Dementsprechend liegt es nahe,  $\ddot{A}(T, w)$  als Relevanzwert für das Dokument  $w$  einzuführen und das Themengebiet  $T$  könnte zum Beispiel als die Menge aller

Dokumente  $w \in S_\sigma$  definiert werden, deren Relevanzwert einen Schwellenwert übertrifft; alle verbleibenden Dokumente sind als nicht-relevant einzustufen. In [BH] wird vorgeschlagen, in der iterativen Authority-Hub Gewichtung die Relevanzwerte mit aufzunehmen. Dies führt auf die Setzungen

$$H_u = \sum_{w, (u,w) \in E_\sigma} h(u, w) \cdot \ddot{A}(T, w) \cdot A_w \quad \text{und} \quad A_u = \sum_{w, (w,u) \in E_\sigma} a(w, u) \cdot \ddot{A}(T, w) \cdot H_w.$$

Weitere Varianten, wie die Eliminierung von nicht-relevanten Dokumenten oder die Eliminierung nur von Dokumenten mit hohem Authority- oder Hub-Gewicht, werden in [BH] beschrieben.

Dieser Ansatz definiert das Themengebiet  $T$  mehr oder minder durch einen Mehrheitsentscheid. Da zudem auch die Bestimmung der Ähnlichkeitswerte aufwändig ist, ist der beträchtliche Zeitaufwand ein großer Nachteil dieses Verfahrens.

## 2.4 Meta-Suchmaschinen

Eine Meta-Suchmaschine gibt eine Anfrage an mehrere Suchmaschinen weiter und berechnet, ausgehend von den Reihenfolgen der einzelnen Suchmaschinen, eine Reihenfolge von hoffentlich relevanten Dokumenten. Eine Meta-Suchmaschine muss damit das folgende *Integrationsproblem* lösen.

Gegeben sind Teilmengen  $T_1, \dots, T_n \subseteq U$  eines Universums  $U$  sowie Ordnungen  $<_1, \dots, <_n$ , wobei  $<_i$  eine vollständige Ordnung auf der Teilmenge  $T_i$  ist. Es ist eine möglichst „gute“ vollständige Ordnung für die Teilmenge  $T = \bigcup_{i=1}^n T_i$  zu bestimmen.

Im Kontext einer Meta-Suchmaschine entspricht das Universum  $U$  der Menge aller Dokumente, die Teilmenge  $T_i \subseteq U$  den von der  $i$ ten Suchmaschine ausgegebenen, potentiell informativen Dokumenten und die vollständige Ordnung  $<_i$  der ausgegebenen Reihenfolge der Dokumente in  $T_i$ .

Das Integrationsproblem kann natürlich auch für konventionelle Suchmaschinen angewandt werden, um verschiedene interne Reihenfolgen zu „integrieren“. Weitere Anwendungen bestehen in Auswahlproblemen, wenn also Objekte unter mehreren Auswahlkriterien auszuwählen sind: Betrachten wir zum Beispiel ein ideales Flug-Reservierungssystem, das dem Anwender erlaubt, seine Präferenzen im Hinblick auf Preis, Reisezeit, Reiselänge, Anzahl der Zwischenstops, Wahl von Fenster- oder Gangsitzen, Frequent-Flier Optionen und Ticket-Rückgaberecht zu nennen. Während das Reservierungssystem für jedes der einzelnen Kriterien eine Reihenfolge der jeweiligen Flugmöglichkeiten ohne Probleme berechnen kann, ist die Berechnung einer für den Anwender zufriedenstellende Reihenfolge im Hinblick auf alle Kriterien die wirkliche Herausforderung.

Zwar ist das Integrationsproblem wegen seiner vielfältigen Anwendungen sehr interessant, doch fehlt eine Erklärung, was denn eine gute vollständige Ordnung ist. Wir werden sogar im nächsten Abschnitt mit Hilfe des Unmöglichkeitssatzes von Arrow sehen, dass eine einfache, überzeugende Erklärung nicht zu erwarten ist.

In Abschnitt 2.4.2 führen wir die Kendall-Distanz ein und wenden Heuristiken zur Berechnungen von Ordnungen mit kleiner Kendall-Distanz zu vorgegebenen Ordnungen  $<_1, \dots, <_n$  an. Die berechneten Ordnungen können experimentell in vielen Fällen überzeugen.

### 2.4.1 Der Unmöglichkeitssatz von Arrow

Wir nehmen in diesem Abschnitt  $T_1 = \dots = T_n = U$  an.

**Definition 2.6** Ein Präferenzen-Funktional  $P$  ordnet jedem Vektor  $(\langle_1, \dots, \langle_n)$  von  $n$  Präferenzrelationen eine Präferenzrelation  $<$  zu.

-  $P$  respektiert *Einstimmigkeit*, falls für alle  $x, y \in U$  :

$$x \langle_i y \text{ für alle } i \in \{1, \dots, n\} \Rightarrow x \langle y.$$

-  $P$  ist *unabhängig von irrelevanten Alternativen*, wenn für alle  $x, y \in U$  nur die  $n$  individuellen Vergleiche zwischen  $x$  und  $y$  über die Präferenz  $x \langle y$  entscheiden.

- Das Funktional  $D_i(\langle_1, \dots, \langle_n) = \langle_i$  heißt *Diktatur-Funktional*.

#### Satz 2.7 Der Unmöglichkeitssatz von Arrow

*Es gelte  $|U| \geq 3$ . Wenn  $P$  ein Präferenzen-Funktional ist, das unabhängig von irrelevanten Alternativen ist und Einstimmigkeit respektiert, dann ist  $P$  ein Diktatur-Funktional.*

Wir überzeugen uns zuerst an einigen Beispielen, dass die Konstruktion eines Präferenzen-Funktional nicht gelingt, wenn Unabhängigkeit von irrelevanten Alternativen und Respektierung der Einstimmigkeit gefordert wird. Der Fall  $|U| = 2$  ist einfach.

**Beispiel 2.1** Wenn  $U = \{x, y\}$ , dann gibt es nur zwei vollständige Ordnungen. Das *Demokratie-Funktional* wählt in diesem Fall die vollständige Ordnung nach dem Mehrheitsprinzip. Bei Stimmgleichheit wird  $x \langle y$  gesetzt. Das *Demokratie-Funktional* respektiert Einstimmigkeit und ist unabhängig von irrelevanten Alternativen.

Schon für  $|U| = 3$  ändert sich die Situation entscheidend, denn es gibt keine „überall gerechten“ Funktionale. Betrachten wir zum Beispiel die Mehrheitsabstimmung auf Paaren:

Definiere die globale Präferenz  $x \langle y$ , falls eine Mehrheit der individuellen Präferenzen  $y$  gegenüber  $x$  bevorzugt.

Hier macht die Transitivität Ärger, denn aus  $x \langle_1 y \langle_1 z$ ,  $y \langle_2 z \langle_2 x$  und  $z \langle_3 x \langle_3 y$  erhalten wir die zyklischen globalen Präferenzen  $x \langle y$ ,  $y \langle z$  und  $z \langle x$ .

**Beweis des Unmöglichkeitssatzes von Arrow:** Wir bezeichnen die individuellen Präferenzen mit  $\langle_1, \dots, \langle_n$  und die vom Präferenzen-Funktional  $P$  bestimmte Präferenzrelation mit  $<$ . Wir setzen voraus, dass  $P$  Einstimmigkeit respektiert und unabhängig von irrelevanten Alternativen ist. Wir müssen zeigen, dass  $P$  ein Diktatur-Funktional ist.

Sei  $S$  eine Teilmenge von  $\{1, \dots, n\}$ . Wir sagen für  $x, y \in U$ , dass  $S$  den *Ausschlag für  $y$  gegenüber  $x$*  gibt, falls stets

$$(\wedge_{i \in S} (x \langle_i y)) \wedge (\wedge_{i \notin S} (y \langle_i x)) \Rightarrow x \langle y$$

gilt: Wenn also  $x \langle_i y$  für alle  $i \in S$  und wenn  $y \langle_i x$  für alle  $i \in \bar{S}$  gilt, dann folgt  $x \langle y$ . Beachte auch, dass  $S$  sowohl den Ausschlag für  $y$  gegenüber  $x$  wie auch für  $x$  gegenüber  $y$  geben kann!

Angenommen, es gilt  $x \langle_i y$  gilt für alle  $i \in S$  wie auch  $y \langle_i x$  für alle  $i \notin S$ . Da die globale Präferenz  $<$  unabhängig von irrelevanten Alternativen ist, hängt die globale Präferenz zwischen  $x$  und  $y$  nur von den individuellen Präferenzen zwischen  $x$  und  $y$  ab. Also ist entweder  $x \langle y$  oder  $y \langle x$  unabhängig von den anderen Elementen  $z \in U$  ( $x \neq z \neq y$ ) und als Konsequenz:

Entweder ist  $S$  ausschlaggebend für  $y$  gegenüber  $x$  oder  $\bar{S}$  ist ausschlaggebend für  $x$  gegenüber  $y$ .

**Behauptung 2.1**  $S$  gebe den Ausschlag für  $y$  gegenüber  $x$ .

- (a) Sei  $z \in U$  mit  $x \neq z \neq y$  beliebig. Dann gibt  $S$  den Ausschlag für  $y$  gegenüber  $z$  wie auch für  $z$  gegenüber  $x$ .
- (b) Seien  $u, v \in U$  mit  $u \neq v$  beliebig gewählt. Dann gibt  $S$  den Ausschlag für  $u$  gegenüber  $v$ .

**Beweis (a):** Wir wissen, dass  $S$  den Ausschlag für  $y$  gegenüber  $x$  gibt.

Wir nehmen an, dass  $z <_i y$  für alle  $i \in S$  und dass  $y <_i z$  für alle  $i \notin S$  gilt. Wir müssen zeigen, dass dann  $z < y$  folgt. Für den Nachweis nutzen wir die Unabhängigkeit von irrelevanten Alternativen aus: Die Größenbeziehung  $z < y$  ist unabhängig von  $x$ . Wir können deshalb annehmen, dass  $z <_i x <_i y$  für jedes  $i \in S$  und dass  $y <_i z <_i x$  für jedes  $i \notin S$  gilt. Nach Annahme können wir  $x < y$  folgern. Aber die globale Präferenz respektiert Einstimmigkeit und  $z < x$  folgt, da alle Spieler  $x$  gegenüber  $z$  bevorzugen. Die Behauptung  $z < y$  ergibt sich somit aus der Transitivität.

Als Nächstes nehmen wir an, dass  $x <_i z$  für alle  $i \in S$  und dass  $z <_i x$  für alle  $i \notin S$  gilt. Wir müssen zeigen, dass dann  $x < z$  folgt. Wie oben schalten wir ein irrelevantes Ergebnis, und zwar diesmal  $y$ , dazwischen: Wir nehmen an, dass  $x <_i y <_i z$  für alle  $i \in S$  und  $y <_i z <_i x$  für alle  $i \notin S$  gilt. Wir erhalten  $x < y$ , da  $S$  den Ausschlag für  $y$  über  $x$  gibt, und  $y < z$  wegen vorliegender Einstimmigkeit. Also folgt die Behauptung  $x < z$  aus der Transitivität.

(b) Wir setzen  $z = v$  und erhalten nach Teil (a), dass  $S$  den Ausschlag für  $v$  gegenüber  $x$  gibt. Wir setzen  $z = u$  und wenden wieder Teil (a) an: Diesmal gibt  $S$  den Ausschlag für  $v$  gegenüber  $u$ .  $\square$

Wenn also eine Menge  $S$  den Ausschlag für  $x$  gegenüber  $y$  gibt, dann ist sie ausschlaggebend für alle möglichen Paare  $u, v$  und wir nennen  $S$  *ausschlaggebend*. Aber entweder gibt  $S$  den Ausschlag für  $y$  gegenüber  $x$  oder  $\bar{S}$  gibt den Ausschlag für  $x$  gegenüber  $y$  und als Konsequenz:

Entweder ist  $S$  ausschlaggebend oder  $\bar{S}$  ist ausschlaggebend.

**Behauptung 2.2** (a) Wenn  $S$  und  $T$  ausschlaggebend sind, dann ist auch  $S \cap T$  ausschlaggebend.

- (b) Wenn  $S \cup T$  ausschlaggebend ist und wenn  $S$  und  $T$  disjunkt sind, dann ist  $S$  oder  $T$  ausschlaggebend.

**Beweis (b):** Für  $x, y, z \in U$  definieren wir die individuellen Präferenzen so, dass

$$S = \{i \mid x <_i y <_i z\} \cup \{i \mid y <_i z <_i x\} \quad \text{und} \quad T = \{i \mid z <_i x <_i y\} \cup \{i \mid y <_i z <_i x\}.$$

Schließlich fordern wir  $x <_i z <_i y$  für die restlichen Spieler und erhalten  $S \cap T = \{i \mid y <_i z <_i x\}$ . Da  $S$  (bzw.  $T$ ) ausschlaggebend ist, folgt  $y < z$  (bzw.  $z < x$ ). Also ergibt sich  $y < x$  aus der Transitivität. Aber dann ist  $S \cap T$  ausschlaggebend, denn  $S \cap T = \{i \mid y <_i z <_i x\}$ .

(b) Angenommen, weder  $S$  noch  $T$  sind ausschlaggebend. Dann sind  $\bar{S}$  und  $\bar{T}$  ausschlaggebend und nach Teil (a) ist  $\bar{S} \cap \bar{T} = \overline{S \cup T}$  ausschlaggebend. Aber  $S \cup T$  ist nach Annahme ausschlaggebend und wir haben einen Widerspruch erhalten.  $\square$

Offensichtlich ist  $\{1, \dots, n\}$  ausschlaggebend, denn die globale Präferenz respektiert Einstimmigkeit. Mit Teil (a) der letzten Behauptung ist  $\{1\}$  ausschlaggebend oder  $\{2, \dots, n\}$  ist ausschlaggebend. Im ersten Fall haben wir nachgewiesen, dass  $P$  ein Diktatur-Funktional ist; ansonsten ist  $\{2, \dots, n\}$  ausschlaggebend und wir iterieren unser Vorgehen solange, bis wir eine Einermenge als ausschlaggebend nachgewiesen haben  $\square$

### 2.4.2 Die Kendall-Distanz

Nach dem Unmöglichkeitssatz von Arrow ist eine eindeutige, überzeugende Definition einer besten vollständigen Ordnung nicht zu erwarten. Wir besprechen eine Reihe bisher vorgeschlagener Definitionen und betrachten dazu zuerst *Borda's Regel*:

Für jedes  $x \in U$  wird  $\text{Rang}_i(x) = |\{z \in U \mid z \leq_i x\}|$ , der Rang von  $x$  bezüglich  $<_i$ , bestimmt. Danach werden die Ergebnisse in der globalen Präferenz gemäß steigender Rangsumme  $\sum_{i=1}^n \text{Rang}_i(x)$  angeordnet.

Borda's Regel kann in Zeit  $O(n \cdot |U|)$  berechnet werden, aber eine Berechnung für partielle Reihenfolgen  $<_1, \dots, <_n$ , also für Reihenfolgen mit  $T_i \subseteq U$  ist unklar. Borda's Regel respektiert Einstimmigkeit aber sie muss nach dem Satz von Arrow die Unabhängigkeit von irrelevanten Alternativen verletzen. (Rangunterschiede sind in Borda's Regel entscheidend und Rangunterschiede werden durch „Dritte“ hervorgerufen.) Borda's Regel verletzt auch die *abgeschwächte Demokratie-Eigenschaft*:

Wenn für irgendeine Teilmenge  $S \subseteq U$  und für alle  $y \in S$  und  $x \in \bar{S}$  stets eine Mehrheit der Ordnungen  $y$  gegenüber  $x$  bevorzugt, dann ist  $x < y$ .

Diese Eigenschaft ist wichtig, um Spamming erfolgreich begegnen zu können: Wenn eine Mehrheit von Suchmaschinen Spamming-Versuche erkennt, dann kann selbst ein hohes Ranking weniger Suchmaschinen kompensiert werden.

---

#### Aufgabe 18

Wir haben eine endliche Menge  $I$  von Wählern und eine endliche Menge  $U$  von Alternativen. Jeder Wähler  $i$  bestimmt seine Präferenz (oder Reihenfolge)  $<_i$  auf  $U$ . Ein Präferenzen-Funktional  $P$  ist

- (i) *symmetrisch*, wenn die berechnete Präferenz nicht von der Reihenfolge der individuellen Präferenzen abhängt. (Es gilt also  $P(<_1, \dots, <_n) = P(<_{\pi(1)}, \dots, <_{\pi(n)})$  für jede Permutation  $\pi$ . Zusätzlich muss für je zwei Alternativen  $x$  und  $y$  eine Vertauschung von  $x$  und  $y$  in jeder Reihenfolge zu einer Vertauschung in der berechneten Präferenz führen.
  - (iii) *monoton*, wenn für jedes  $x, y \in U$  mit  $x < y$  die Beziehung  $x < y$  weiterhin gilt, falls  $x$  und  $y$  in irgendeiner Reihenfolge  $<_i$  vertauscht werden, so dass jetzt  $x$  eine kleinere Priorität als  $y$  erhält.
- (a) Es sei  $|U| = 2$ . Zeige, dass das Demokratie-Funktional das *einzigste* Präferenzen-Funktional ist, das sowohl symmetrisch wie auch monoton ist.
- (b) Es gelte  $|U| \geq 3$ . Ist Borda's Regel symmetrisch? Ist Borda's Regel monoton?
- (c) Zeige, dass Borda's Regel die abgeschwächte Demokratie-Eigenschaft bereits für  $|S| = 1$  verletzt.
- 

Die *Spearman-Distanz*  $d_1(<_1, <_2)$  zwischen zwei vollständigen Ordnungen  $<_1$  und  $<_2$  wird durch

$$d_1(<_1, <_2) = \sum_{x \in U} |\text{Rang}_1(x) - \text{Rang}_2(x)|$$

definiert. Für gegebene vollständige Ordnungen  $<_1, \dots, <_n$  ist dann

$$s(<_1, \dots, <_n; <) = \sum_{i=1}^n d_1(<_i, <)$$

die von der vollständigen Ordnung  $<$  erreichte Distanz.

**Lemma 2.8** *Eine für die Spearman-Distanz optimale vollständige Ordnung kann effizient berechnet werden.*

**Beweis:** Wir stellen einen vollständigen bipartiten Graphen mit den Knotenmengen  $V_1 = U$  und  $V_2 = \{1, \dots, |U|\}$  auf. Für ein Element  $x \in V_1$  und eine „Position“  $p \in V_2$  wird die Kante  $\{x, p\}$  mit dem Gewicht  $\sum_{i=1}^n |\text{Rang}_i(x) - p|$  versehen.

Ein perfektes Matching, also eine Teilmenge  $E$  der Kantenmenge, so dass jeder Knoten aus  $V_1$  wie auch jeder Knoten aus  $V_2$  Endpunkt von genau einer Kante aus  $E$  ist, definiert eine vollständige Ordnung: Wenn  $\{x, p\} \in E$ , dann definieren wir  $p$  als den Rang von  $x$  in  $<$ . Umgekehrt definiert eine vollständige Ordnung das perfekte Matching  $\{\{x, \text{Rang}(x)\} \mid x \in V_1\}$  und wir erhalten eine eindeutige Entsprechung von Matchings und vollständigen Ordnungen. Jetzt brauchen wir nur noch zu beobachten, dass  $s(<_1, \dots, <_n; <)$  das Kantengewicht des Matchings von  $<$  ist. Also ist ein perfektes Matching von minimalem Gewicht eine optimale Shearman Lösung. Die Behauptung folgt, da perfekte Matchings von minimalem Gewicht effizient bestimmt werden können.  $\square$

Die Berechnung optimaler Reihenfolge für partielle Reihenfolgen  $<_1, \dots, <_n$  führt allerdings wieder auf ein  $NP$ -vollständiges Problem, wenn wir für die Berechnung der Distanz  $d_1(<_i, <)$  nur die Ränge von Elementen aus  $T_i$  heranziehen. Trotzdem lassen sich gute Ergebnisse nach einer geringfügigen Modifikation des Algorithmus aus Lemma 2.8 erzielen: Für die neuen Setzungen  $V_1 = T$  (mit  $T = \bigcup_{i=1}^n T_i$ ) und  $V_2 = \{1, \dots, |T|\}$  wird die neue Gewichtung  $\sum_{i=1}^n |\text{Rang}_i(x)/|T_i| - p/|T||$  der Kante  $\{x, p\}$  verwandt.

Die *Kendall-Distanz*  $d_2(<_1, <_2)$  wird durch

$$d_2(<_1, <_2) = |\{(x, y) \in U \mid x <_1 y \text{ und } y <_2 x\}|$$

definiert und ist damit die minimale Anzahl von Transpositionen benachbarter Elemente, um die Reihenfolge  $<_1$  in die Reihenfolge  $<_2$  zu verwandeln. Auch hier wird die Distanz einer vollständigen Ordnung  $<$  zu gegebenen vollständigen Ordnungen  $<_1, \dots, <_n$  als Summe der Distanzen, also durch

$$k(<_1, \dots, <_n; <) = \sum_{i=1}^n d_2(<_i, <)$$

festgelegt. Für partielle Reihenfolgen  $<_1, \dots, <_n$  beschränkt man sich zur Berechnung von  $d_2(<_i, <)$  auf die Elemente aus  $T_i$ . Leider stellt sich die Berechnung optimaler Reihenfolgen schon für  $n = 4$  und selbst für vollständige Reihenfolgen als ein  $NP$ -vollständiges Problem heraus. Allerdings besteht ein enger Zusammenhang zwischen der Spearman- und der Kendall-Distanz:

---

**Aufgabe 19**

Zeige, dass stets  $d_2(<_1, <_2) \leq d_1(<_1, <_2) \leq 2 \cdot d_2(<_1, <_2)$  für vollständige Reihenfolgen gilt. Optimale Lösungen für die Kendall-Distanz können also effizient bis auf den Faktor zwei approximiert werden.

---

Wir zeigen, dass Kendall-optimale Ordnungen  $<$  die abgeschwächte Demokratie-Eigenschaft besitzen. Da eine effiziente Berechnung Kendall-optimaler Lösungen nicht wahrscheinlich ist, müssen wir aber auf Approximationen zurückgreifen und damit besteht die Gefahr, dass die abgeschwächte Demokratie-Eigenschaft verletzt ist. Eine Wiederherstellung ist aber selbst für partielle Reihenfolgen einfach.

**Lemma 2.9** Die Reihenfolgen  $\langle_1, \dots, \langle_n$  und  $\langle$  seien vorgegeben. Dann kann eine Reihenfolge  $\langle^*$  in Zeit  $O(n \cdot |\bigcup_{i=1}^n T_i|^2)$  bestimmt werden, so dass

- $\langle^*$  die abgeschwächte Demokratie-Eigenschaft besitzt und
- $k(\langle_1, \dots, \langle_n, \langle^*) \leq k(\langle_1, \dots, \langle_n, \langle)$  gilt.

**Beweis:** O.B.d.A. sei  $U = \{1, \dots, |U|\}$  und es gelte  $1 > 2 > \dots > |U|$ . Wir bestimmen  $\langle^*$  iterativ. Wenn wir  $\langle^*$  schon auf den Elementen  $1, \dots, k$  definiert haben, dann legen wir  $\langle^*$  als Nächstes auf den Elementen  $1, \dots, k, k+1$  fest. Wir versuchen zuerst  $k+1$  als kleinstes der jetzt  $k+1$  vielen Elemente festzulegen, vertauschen aber  $k+1$  mit dem  $k$ -kleinsten Element, wenn eine Mehrheit der Reihenfolgen  $\langle_1, \dots, \langle_n$  dies verlangt. Dieser Vertauschungsprozess wird solange wiederholt, bis Element  $k+1$  zur „Ruhe kommt“.

Wir beachten zuerst, dass  $\langle^*$  mindestens so gut wie  $\langle$  ist, da kein Vertauschungsschritt die  $k$ -Distanz erhöht. Wir verifizieren die abgeschwächte Demokratie-Eigenschaft als nächstes. Offensichtlich ist  $\langle^*$  bezüglich der Distanzmessung  $k(\langle_1, \dots, \langle_n, \langle^*)$  im folgenden Sinne *lokal optimal*: Wenn wir irgendein Paar von in  $\langle^*$  aufeinanderfolgenden Elementen vertauschen, dann wird die Distanz  $k$  nicht abnehmen.

Angenommen  $\langle^*$  erfüllt die abgeschwächte Demokratie-Eigenschaft nicht. Dann gibt es eine Teilmenge  $S \subseteq \{1, \dots, n\}$ , so dass von den meisten Ordnungen stets  $s \in S$  gegenüber  $t \notin S$  bevorzugt wird. Allerdings gibt es auch Elemente  $u \in S, v \in T$  mit  $u < v$ . Wir nehmen an, dass das Paar  $(u, v)$ , bezüglich  $\langle$ , unter allen Gegenbeispielen für die abgeschwächte Demokratie-Eigenschaft unter der Ordnung  $\langle$  die geringste Distanz besitzt.

Sei  $w$  der unmittelbare Nachfolger von  $v$  bezüglich  $\langle$ , d.h.  $w$  ist das bezüglich  $\langle$  größte Element kleiner als  $v$ . Wenn  $w = u$ , dann ist  $\langle^*$  nicht lokal maximal, da eine Vertauschung von  $v$  und  $u$  die Gesamtdistanz verringert. Also ist  $v > w > u$  und  $w$  muss zur Menge  $S$  gehören, da  $(u, w)$  ansonsten ein Gegenbeispiel geringerer Distanz ist. Aber auch dies ist nicht möglich, da dann  $(w, v)$  ein Gegenbeispiel von benachbarten Elementen ist.  $\square$

Die Berechnung von  $\langle^*$  benötigt  $\Theta(n \cdot |U|^2)$  Operationen im Worst-Case, aber wir können die Rechenzeit signifikant verbessern, wenn wir nur irgendein lokales Optimum finden möchten. Dazu definieren wir einen gerichteten Graphen  $G = (U, E)$  mit Knotenmenge  $U$  und fügen Kanten  $(y, x)$  ein, wann immer eine Mehrheit der Ordnungen  $\langle_i$  mit  $x, y \in T_i$  das Element  $y$  gegenüber  $x$  bevorzugt; bei einem Unentschieden setzen wir die beiden Kanten  $(x, y)$  und  $(y, x)$  ein. Beachte, dass lokal optimale Ordnungen und Hamilton'sche Wege einander entsprechen.

---

#### Aufgabe 20

Zeige, dass eine lokal optimale Ordnung in Zeit  $O(k \cdot |T| \cdot \log_2 |T|)$  gefunden werden kann.

---

#### Aufgabe 21

Die Ordnungen  $\langle_1, \dots, \langle_n$  und  $\langle$  seien vorgegeben. Wir nennen  $\langle^*$  konsistent mit  $\langle_1, \dots, \langle_n$  und  $\langle$ , falls wann immer  $x \langle^* y$  auch  $x \langle y$  gilt, oder aber eine Mehrheit der Ordnungen  $\langle_i$  mit  $x, y \in T_i$  bevorzugt das Element  $y$  gegenüber  $x$ .

(a) Zeige, dass  $\langle^*$  mindestens so gut wie  $\langle$  bezüglich  $k$  ist.

(b) Wenn wir die Ordnungen auf die  $m$  höchsten (gemäß  $\langle$ ) Elemente einschränken, dann ist  $\langle^*$  eindeutig bestimmt, falls  $\langle^*$  lokal optimal und konsistent mit  $\langle_1, \dots, \langle_n$  und  $\langle$  ist.

---

Bisher werden vor Allem Varianten von Borda's Regel zur Bestimmung guter Ordnungen für die Kendall-Distanz eingesetzt. Wir haben aber bereits mit dem Verfahren von Lemma 2.8 eine weitere Heuristik kennengelernt. Da die Spearman-Distanz die Kendall-Distanz bis auf den Faktor zwei approximiert, überrascht es nicht, dass im Vergleich zur Borda's Regel

experimentell eine weitaus bessere Approximation beobachtet wurde. Wir lernen jetzt eine dritte, einfachere aber in experimentellen Tests mindestens so erfolgreiche Heuristik kennen.

**Algorithmus 2.10 Der Markoff-Ketten Ansatz.**

- (1) Bestimme  $T = \bigcup_{i=1}^n T_i$ .
- (2) Konstruiere eine Markoff-Kette  $(G, P)$  mit Zustandsmenge  $T$  wie folgt. Für  $x \in T$  sei

$$\text{Größer}(x) = \{y \mid x <_i y \text{ für die meisten Ordnungen } <_i \text{ mit } x, y \in T_i\}.$$

Dann setze  $P[x, y] = \frac{1}{|\text{Größer}(x)|}$ , falls  $y \in \text{Größer}(x)$ , und  $P[x, y] = 0$  sonst.

- (3) Bestimme die stationäre Verteilung  $\pi$  von  $(G, P)$ .
- (4) Berechne eine vollständige Ordnung  $<$  für  $\pi$ .

Wir müssen die Berechnung der Ordnung  $<$  für die stationäre Verteilung  $\pi$  präzisieren. Der gerichtete Graph  $G = (V, E)$  zerfällt in starke Zusammenhangskomponenten<sup>4</sup>. Sei  $\mathcal{X}$  die Menge der starken Zusammenhangskomponenten. Wir bauen einen weiteren gerichteten Graph  $G^* = (\mathcal{X}, E^*)$ , wobei wir eine Kante  $(X_1, X_2)$  für  $X_1, X_2 \in \mathcal{X}$  einsetzen, wenn  $G$  eine Kante  $(x_1, x_2)$  mit  $x_1 \in X_1$  und  $x_2 \in X_2$  besitzt.

---

**Aufgabe 22**

Zeige, dass  $G^*$  kreisfrei ist.

---

Wenn  $x \in X$  und wenn  $X$  keine Senke ist, dann ist die Wahrscheinlichkeit den Knoten  $x$  beliebig oft zu erreichen gleich Null und es gilt  $\pi(x) = 0$ . Wir bestimmen deshalb zuerst  $\pi$  nur für solche Knoten, die zu den Senken von  $G^*$  gehören, und fordern  $x_1 < x_2$ , wann immer  $\pi(x_1) < \pi(x_2)$  gilt.

Als Nächstes entfernen wir alle Senken aus  $G^*$  und alle Knoten aus  $G$ , die zu den Senken von  $G^*$  gehören. Wir bilden gemäß Algorithmus 2.10 eine Markoff-Kette auf den verbleibenden Knoten und wiederholen das obige Verfahren, wobei die neuen Knoten in  $<$  unterhalb den alten Knoten angesiedelt werden.

---

**Offenes Problem 1**

Bestimme den Approximationsfaktor des Markoff-Ketten Ansatzes.

---

Sowohl für die Spearman-Distanz wie auch für die Kendall-Distanz kann eine Gewichtung der individuellen Ordnungen  $<_1, \dots, <_n$  eingeführt werden, um die Voraussagekraft der jeweiligen Reihenfolgen auszudrücken. Die obigen Ergebnisse bleiben weiterhin bestehen.

## 2.5 Das Semantische Netz

Nach Berners-Lee, einem Pionier des World-Wide-Web, ist das Semantic Web eine Erweiterung des herkömmlichen Webs, in der Informationen mit Maschinen-verständlichen Bedeutungen versehen werden, um die Semantik der Inhalte formal festlegen.

---

<sup>4</sup>Ein gerichteter Graph heißt stark zusammenhängend, falls je zwei Knoten des Graphen durch einen Weg miteinander verbunden sind. Ein Teilgraph  $H = (V', E')$  mit  $E' \subseteq E$  von  $G$  heißt eine starke Zusammenhangskomponente von  $G$ , falls  $H$  stark zusammenhängend ist und kein Teilgraph, der  $H$  echt enthält, stark zusammenhängend ist.

Die Grundidee ist die Implementierung eines verteilten semantischen Netzes, in der die Annotation der HTML/XML-Seiten eine wichtige Rolle spielt. Zum einen wird die Bedeutung von WWW-Links durch die Annotation erklärt, zum anderen wird mit der Annotation Reasoning möglich: Besagt die Annotation eines Dokuments zum Beispiel, dass es sich mit „Fußball“ beschäftigt und geht aus der zugrundeliegenden Ontologie hervor, dass der Begriff „Fußball“ eine „Sportart“ darstellt, dann kann geschlossen werden, dass es im Dokument also auch um Sport geht, obwohl dies nicht explizit in den Metadaten auftaucht. Bei entsprechender Qualität der Annotation lässt sich ein hoher Grad automatischer Verarbeitung erreichen. So wäre es zum Beispiel sogar denkbar, dass eine Suchmaschine im Semantischen Netz Anfragen der Art „Wie viele Tore hat Fußballer X im Jahre 1998 geschossen?“ direkt beantworten kann. Allerdings ist es unklar, ob sich das semantische Web durchsetzen wird: Ein überzeugender Standard hat sich bisher nicht etablieren können. Auch wird die zusätzliche Annotation des Anwenders nicht unmittelbar belohnt und es stellt sich die Frage, ob viele Anwender diese zusätzliche Arbeit auf sich nehmen werden. Das Konzept des semantischen Netzes ist aber sehr attraktiv und, bei einer Einschränkung auf objektive semantische Daten, werden Suchmaschinen profitieren können.

## 2.6 Zusammenfassung

Wir haben den *Random Surfer Ansatz* von Google sowie den *Hubs-Authorities Ansatz* von Kleinberg vorgestellt. Beide Ansätze bewerten Dokumente durch einen Peer-Review mit Hilfe der *Connectivity-Analyse*. Um erfolgreich zu sein, müssen deshalb die Annahmen der Connectivity-Analyse zutreffen: Wenn ein Dokument  $A$  auf Dokument  $B$  zeigt, dann

- gibt es eine inhaltliche Beziehung zwischen den beiden Dokumenten und
- der Autor des Dokuments  $A$  hält Dokument  $B$  für wertvoll.

**Google** berechnet den Page-Rank eines Dokuments  $w$  als eine Anfrage-unabhängige Bewertung: Der Page-Rank  $\text{pr}(w)$  ist hoch, wenn viele Dokumente  $u$  mit hohem Page-Rank  $\text{pr}(u)$  auf das Dokument  $w$  zeigen. Eine Formalisierung des Page-Rank Konzepts führt dann auf

$$\text{pr} = d \cdot p + (1 - d) \cdot \text{pr} \cdot P, \quad (2.5)$$

wobei der Vektor  $p$  durch  $p_w = \frac{1}{n}$  definiert ist. Der mit einer (im Vergleich zu 1) kleinen Konstante  $d$  gewichtete Vektor  $p$  garantiert, dass der Page-Rank mit der stationären Verteilung auf dem Webgraphen übereinstimmt, wenn wir die Übergangswahrscheinlichkeiten

$$P'[u, w] = \begin{cases} \frac{d}{n} + \frac{1-d}{d_u} & (u, w) \text{ ist ein Hyperlink} \\ \frac{d}{n} & \text{sonst.} \end{cases}$$

für den Übergang von Dokument  $u$  auf Dokument  $w$  wählen: Die Forderung (2.5) besitzt eine eindeutige Lösung, wenn wir zusätzlich fordern, dass der Page-Rank Vektor eine Verteilung ist. Als Konsequenz stimmt der Page-Rank  $\text{pr}_w$  mit der relativen Häufigkeit überein, dass ein Random Surfer das Dokument  $w$  besucht.

Ein wesentlicher Nachteil ist die Unabhängigkeit des Page-Ranks von der Anfrage. Wir haben deshalb die Möglichkeit benutzt, den Vektor  $p$  in (2.5) für ein vorgegebenes Themengebiet  $T$

maßzuschneidern. Wenn ein Crawler das Dokument  $w$  besucht und die Kompetenz  $p_w^T$  für das Themengebiet feststellt, dann können wir

$$\text{pr}^{(i)} = d \cdot p^T + (1 - d) \cdot \text{pr}^{(i)} \cdot P, \quad (2.6)$$

als den Page-Rank für Themengebiet  $T$  einführen. Dabei genügt eine text-basierte Kompetenzbewertung  $p_w^T$  von  $w$ , da wir ja weiterhin eine Connectivity-Analyse ausführen.

Der **HITS-Algorithmus** berechnet simultan Hubs (Dokumente mit „guten“ Links) und Authorities (Dokumente von hoher Qualität). In der Berechnung erhält ein Dokument, das auf viele Dokumente mit hohem Authority-Gewicht zeigt, ein hohes Hub-Gewicht

$$H_u = \sum_{w, (u,w) \in E_\sigma} A_w,$$

während ein Dokument, auf das viele Dokumente mit hohem Hub-Gewicht zeigen, ein hohes Authority-Gewicht

$$A_u = \sum_{w, (w,u) \in E_\sigma} H_w$$

erhält. In Anwendungen konvergieren die Hub- und Authorities-Vektoren nach nur relativ wenigen Iterationen gegen die größten Eigenvektoren von  $M \cdot M^T$  bzw. von  $M^T \cdot M$ , wobei  $M$  die Adjazenzmatrix des Webgraphen ist.

Schließlich haben wir möglicherweise zu allgemeine Hubs oder Authorities durch eine Vorverarbeitung ausgeschlossen. Dazu haben wir das Themengebiet der Anfrage abstrakt definiert und die „Ähnlichkeit“ mit dem Themengebiet bestimmt.

Das **Integrationsproblem** ist für den Entwurf von Metasuchmaschinen und allgemeiner für die Erstellung von Reihenfolgen im Hinblick auf mehrere Kriterien wesentlich. Wir haben zuerst im Satz von Arrow gesehen, dass die Definition einer zufriedenstellenden Reihenfolge schwierig ist. Da sich eine Metasuchmaschine vorrangig mit Spamming auseinandersetzen muss, haben wir die Kendall-Distanz oder Bubblesort-Distanz

$$d(\langle_1, \langle_2) = |\{(x, y) \in U \mid x \langle_1 y \text{ und } y \langle_2 x\}|$$

zwischen zwei Reihenfolgen  $\langle_1$  und  $\langle_2$  eingeführt und die Qualität einer Reihenfolge  $\langle$  im Hinblick auf die Reihenfolgen oder Kriterien  $\langle_1, \dots, \langle_n$  durch die Distanzsumme

$$k(\langle_1, \dots, \langle_n; \langle) = \sum_{i=1}^n d(\langle_i, \langle)$$

gemessen. Die Bestimmung einer Reihenfolge mit kleinster Distanzsumme führt allerdings auf ein  $NP$ -vollständiges Problem, aber gute Approximationen können zum Beispiel mit Markoff-Ketten effizient berechnet werden. Wenn wir den Vertauschungsalgorithmus aus Lemma 2.9 anwenden, dann erfüllen diese Approximationen auch die abgeschwächte Demokratie-Eigenschaft: Wenn für irgendeine Teilmenge  $S \subseteq \{1, \dots, n\}$  und für alle  $y \in S$  und  $x \in \bar{S}$  stets eine Mehrheit der Reihenfolgen  $y$  gegenüber  $x$  bevorzugt, dann ist  $x \langle y$ . Die abgeschwächte Demokratie-Eigenschaft kann gegen Spamming eingesetzt werden: Wenn eine Mehrheit von Suchmaschinen Spamming-Versuche erkennt, dann kann selbst ein hohes Ranking „verdorbener“ Dokumente (unterstützt nur durch eine Minderheit der Suchmaschinen) kompensiert werden.

## Kapitel 3

# Hashing für Web-Anwendungen

Wir beschreiben zuerst verteilte Hashing-Verfahren, die in File-Sharing Anwendungen für Peer-to-Peer Systeme<sup>1</sup> eine wichtige Rolle spielen.

Als Nächstes stellen wir uns das Problem, alle Paare ähnlicher Objekte in einer großen Menge von Objekte zu bestimmen. Wenn identische Objekte herauszufiltern sind, dann ist konventionelles Hashing eine gute Lösung. Auch die Bestimmung ähnlicher Objekte scheint trivial, da ein Algorithmus mit quadratischer Laufzeit alle Paare inspizieren und auswerten kann. Allerdings können wir uns eine quadratische Laufzeit im Hinblick auf die Größe der Datenmenge nicht leisten, und wir benutzen deshalb *Min-Hashing*, eine Variante des Ähnlichkeitsbewahrenden Hashings.

Sodann speichern wir Mengen speicherplatz-effizient mit Hilfe von *Bloom-Filtern* ab. Bloom-Filter besitzen viele Anwendungen in Netzwerken und zwar besonders dann, wenn Knoten des Netzwerks Mengen relevanter Informationen austauschen müssen: Der Netzwerk-Verkehr wird Dank der Kompression durch Bloom-Filter weit weniger als mit konventionellen Methoden belastet.

### 3.1 Verteiltes Hashing in Peer-to-Peer Netzwerken

Die Musiktatschbörse *Napster* erlaubt den Austausch von MP3-Musikdateien über das Internet und verfolgt dazu eine Mischung des Client-Server und des Peer-to-Peer Ansatzes: Die Napster-Software durchsucht den Kunden-Rechner nach MP3-Dateien und meldet die Ergebnisse an einen zentralen Server, der auch alleinigen Zugriff auf die Angebote und Suchanfragen der anderen Teilnehmer hat (Client-Server Ansatz). Der Server meldet dann als Ergebnis auf eine Anfrage die IP-Adressen der Rechner, die die gesuchte Musikdatei anbieten. Anbieter und Käufer können sich daraufhin direkt miteinander verbinden (Peer-to-Peer Ansatz).

Das *Gnutella-Netzwerk* ist ein vollständig dezentrales Netzwerk ohne zentrale Server. Startet ein Benutzer des Netzwerkes eine Suchanfrage, so wird diese zunächst nur an benachbarte Systeme weitergeleitet. Diese leiten dann ihrerseits die Anfrage an ihre benachbarten Systeme weiter, bis die angeforderte Datei gefunden wurde. Anschließend kann eine direkte Verbindung zwischen suchendem und anbietendem Benutzer für die Datenübertragung hergestellt werden. Gnutella überflutet also das Netz mit der Suchanfrage und wird deshalb nur unzureichend skalieren.

---

<sup>1</sup>In einem Peer-to-Peer-Netzwerk sind, im Gegensatz zu Client-Server Netzen, alle Rechner gleichberechtigt und können sowohl Dienste in Anspruch nehmen als auch Dienste zur Verfügung stellen.

Reine Peer-to-Peer Systeme wie Gnutella sind ausfallsicherer und verteilen die hohe Belastung des Servers auf die Peers. Wir beschreiben das reine Peer-to-Peer System *CHORD*, das sowohl den Zugriff auf Information wie auch die Registrierung neuer Peers oder das Abmelden alter Peers **dezentral** und **effizient** erlaubt.

### 3.1.0.1 Chord

Unser Ziel ist die Definition eines Protokolls, das den effizienten Zugriff auf Information erlaubt, wenn Informationen auf verschiedenste Rechner verteilt ist und wenn Rechner das Netzwerk verlassen bzw. dem Netzwerk beitreten dürfen. Beachte, dass Information durch den Weggang, bzw. den Zugewinn von Rechnern bewegt werden muss und gute Protokolle sollte diese dynamischen Veränderungen ohne großen zusätzlichen Kommunikationsaufwand bewältigen.

Die zentrale algorithmische Idee besteht in dem Konzept konsistenter Hashfunktionen, die sowohl auf die Menge  $\mathcal{R}$  aller möglichen IP-Adressen wie auch auf die Menge  $\mathcal{D}$  aller möglichen Dateien-ID's angewandt werden dürfen. Im Unterschied zu den bisherigen Anwendungen liefern unsere Hashfunktionen jetzt reelle Zahlen in dem Intervall  $[0, 1]$ .

**Definition 3.1** (a) Die Klasse konsistenter Hashfunktionen besteht aus allen Funktionen der Form  $h : \mathcal{R} \cup \mathcal{D} \rightarrow [0, 1]$ .

(b) Sei  $R$  eine Teilmenge von  $\mathcal{R}$ . Wenn  $h(r_1) < h(r_2)$  für zwei Rechner  $r_1, r_2 \in R$  gilt und wenn es keinen Rechner  $r_3 \in R$  mit  $h(r_1) < h(r_3) < h(r_2)$  gibt, dann heisst  $r_2$  der Nachfolger von  $r_1$  und  $r_1$  der Vorgänger von  $r_2$  bezüglich  $h$ .

Wir rechnen modulo 1, d.h. der Rechner mit größtem Hashwert ist Vorgänger des Rechners Rechner mit kleinstem Hashwert. Demgemäß sollte man sich das Intervall  $[0, 1]$  als Kreis vorstellen.

(c) Sei  $R$  eine Teilmenge von  $\mathcal{R}$  und  $D$  eine Teilmenge von  $\mathcal{D}$ . Dann sagen wir, dass der Rechner  $r_2 \in R$  für die Dateimenge  $D' \subseteq D$  zuständig ist, falls  $r_2 \in R$  der Nachfolger von  $r_1$  ist und  $D' = \{d \in D \mid h(r_1) < h(d) < h(r_2)\}$  gilt.

Angenommen die Dateimenge  $D$  ist von den Rechnern in  $R$  abzuspeichern. Dann wird in Chord ein Rechner genau die Dateien speichern, für die er zuständig ist. Wenn jetzt ein Rechner  $r \in R$  auf die Datei  $d \in D$  zugreifen möchte, dann berechnet  $r$  den Hashwert  $h(d)$ . Chord stellt zusätzliche Information, die **Routing-Tabelle**, bereit, so dass  $r$  den für  $d$  zuständigen Rechner in wenigen Schritten lokalisieren kann:

- (1)  $r$  erhält die IP-Adresse seines Vorgängers und seines Nachfolgers.
- (2) Die Rechner sind durch die Hashfunktion auf dem Kreis angeordnet. Für jedes  $k$  erhält  $r$  desweiteren die IP-Adresse  $R_r(k)$  des ersten Rechners mit einem Abstand von mindestens  $2^{-k}$ , d.h. es ist  $h(R_r(k)) \geq h(r) + 2^{-k}$  und für jeden Rechner  $t \in R$  zwischen  $r$  und  $R_r(k)$  ist  $h(t) < h(r) + 2^{-k}$ .

Auf der Suche nach Datei  $d$  bestimmt  $r$  den Wert  $k$ , so dass  $h(R_r(k)) \leq h(d) \leq h(R_r(k-1))$  und sendet seine Anfrage nach  $d$  an den Rechner  $s = R_r(k)$  mit der Bitte bei der Suche zu helfen;  $s$  ist also der Rechner der Routingtabelle von  $r$  mit größtem Hashwert beschränkt durch  $h(d)$ . Rechner  $s$  bestimmt seinerseits den Wert  $l$  mit  $h(R_s(l)) \leq h(d) \leq h(R_s(l-1))$  und kontaktiert  $R_s(l)$ . Dieses Verfahren wird solange wiederholt bis der Vorgänger  $v$  des für

die Datei  $d$  zuständigen Rechners erreicht ist. Rechner  $v$  kontaktiert seinen Nachfolger, der  $d$  dann direkt an den ursprünglich anfragenden Rechner  $r$  verschickt.

Beachte, dass  $r$  und alle auf der Suche nach  $d$  beteiligten Rechner jeweils Rechner  $r'$  mit  $h(r') \leq h(v)$  kontaktieren. Wir behaupten, dass die Hashdistanz zu  $h(v)$  in jedem Suchschritt mindestens halbiert wird.

**Lemma 3.2** *Wenn  $r'$  auf der Suche nach Datei  $d$  den Rechner  $r''$  kontaktiert, dann ist*

$$h(v) - h(r'') \leq \frac{h(v) - h(r')}{2}.$$

**Beweis:** Es gelte  $h(r') + 2^{-k} \leq h(v) < h(r') + 2^{-(k-1)}$ . Dann wird  $r'$  den Rechner  $r''$  aus seiner Routingtabelle mit  $h(r') + 2^{-k} \leq h(r'') \leq h(v)$  kontaktieren. Also folgt

$$h(v) - h(r') = h(v) - h(r'') + h(r'') - h(r') \geq h(v) - h(r'') + 2^{-k} \geq 2 \cdot (h(v) - h(r''))$$

und das war zu zeigen.  $\square$

Wir erwähnen ohne Beweis, dass die Rechner bei zufälliger Wahl der Hashfunktion gleichmäßig mit Dateien belastet werden und dass die Rechner gleichmäßig auf dem Kreis verteilt sind.

**Lemma 3.3** *Sei  $|R| = n$  und  $|D| = m$ . Für hinreichend großes  $\alpha$  gelten die folgenden Aussagen nach zufälliger Wahl einer konsistenten Hashfunktion mit "großer" Wahrscheinlichkeit:*

- (a) *Jeder Rechner ist für höchstens  $\alpha \cdot \frac{m}{n} \cdot \log_2(n + m)$  Dateien zuständig.*
- (b) *Kein Interval der Länge  $\frac{\alpha}{n^2}$  enthält zwei Rechner und kein Interval der Länge  $\frac{\alpha}{m^2}$  enthält zwei Dateien.*
- (c) *Die Suche nach der Datei  $s$  ist nach höchstens  $\alpha \cdot \log_2 n$  Schritten erfolgreich.*

Betrachten wir als Nächstes das Einfügen von Rechnern. Wenn Rechner  $r$  dem System beitreten möchte, dann bittet  $r$  einen beliebigen Rechner des Systems die Suche nach  $h(r)$  durchzuführen. Nach erfolgreicher Suche hat  $r$  seinen Nachfolger  $r'$  gefunden und kann  $r'$  benachrichtigen.  $r'$  übermittelt die IP-Adresse des bisherigen Vorgängers und betrachtet  $r$  als seinen neuen Vorgänger. Danach baut  $r$  seine Routing-Tabelle mit Hilfe seines Nachfolgers  $r'$  auf: Wenn  $r$  auf die Anfrage nach dem fiktiven Hashwert  $h(r) + 2^{-k}$  die IP-Adresse  $r_k$  erhält, dann ist  $R_r(k) = r_k$ .

Warum besteht kein Grund zur Panik, wenn ein Hinzufügen von Rechnern nicht sofort vollständig in den Routing-Tabellen vermerkt wird? Solange der Rechner  $R_r(k)$  nicht entfernt wurde, hilft  $R_r(k)$  weiterhin in der Halbierung der Distanz und das gilt selbst dann, wenn nähere Knoten im Abstand mindestens  $2^{-k}$  zwischenzeitlich eingefügt wurden. Das erwartete Verhalten nach dem Einfügen von Rechnern ist beweisbar "gutmütig":

**Lemma 3.4** *Zu einem gegebenen Zeitpunkt bestehe das Netz aus  $n$  Rechnern und alle Zeiger-Informationen (also Vorgänger, Nachfolger und Distanz  $2^{-k}$  Rechner) seien korrekt. Wenn dann  $n$  weitere Rechner hinzugefügt werden und selbst wenn nur Vorgänger und Nachfolger richtig aktualisiert werden, dann gelingt eine Suche hochwahrscheinlich in Zeit  $O(\log_2 n)$ .*

---

#### Aufgabe 23

Zeige Lemma 3.4. Benutze, dass hochwahrscheinlich höchstens  $O(\log_2 n)$  neue Rechner zwischen zwei alten Rechnern liegen.

---

Wenn sich ein Rechner  $r$  abmeldet, dann ist jetzt der Nachfolger von  $r$  für die Schlüssel von  $r$  zuständig. Desweiteren muss sich  $r$  bei seinem Nachfolger und seinem Vorgänger abmelden. Auch diesmal versucht Chord nicht, die Routing-Tabellen vollständig zu aktualisieren und stattdessen überprüft jeder Rechner seine Routing-Tabelle in periodischen Abständen. Sollte sich während der Suche nach einer Datei herausstellen, dass Rechner  $R_r(k)$  verschwunden ist, dann leidet die Suche unmittelbar:  $r$  muss sich auf die Suche machen und kontaktiert  $R_r(k-1)$  mit dem imaginären Schlüsselwert  $h(r) + 2^{-k}$ ; nach erhaltener Antwort kann die Dateisuche fortgesetzt werden.

---

**Aufgabe 24**

(a) Zeige, dass die folgenden pathologischen Fälle nach Einfügen und Entfernen von Rechnern auftreten können:

- Die Nachfolger-Zeiger definieren eine Liste, die mehrfach über das Intervall  $[0, 1]$  "streicht".
- Die Nachfolger-Zeiger definieren mehrere disjunkte Zyklen.

(b) Wie lassen sich diese Pathologien erkennen?

---

Ein zentrales Problem für Chord ist die Behandlung bössartiger Rechner, ein Problem das zum jetzigen Zeitpunkt noch ungelöst ist. Ein ähnliches Problem ist der Ausfall von Rechnern ohne vorige Abmeldung; dieses Problem ist aber zumindest approximativ durch Datenreplikation über eine fixe Anzahl von Nachfolgern in den Griff zu bekommen. Beachte, dass die zufällige Wahl der Hashfunktion einerseits für eine gleichmäßige Auslastung der Rechner und eine schnelle Suchzeit sorgt; andererseits ist sie aber auch dafür verantwortlich, dass die "Hop-Distanz" zwischen auf dem Kreis benachbarten Rechnern groß sein wird.

Weitere Distributed Hashing Ansätze werden in den Systemen CAN, Pastry und Tapestry verfolgt. Ein Vergleich wird in [BKKMS] durchgeführt.

## 3.2 Ähnlichkeitsbestimmung

Sei  $X$  eine Menge von Objekten. Wir definieren ein Ähnlichkeitsmaß als eine Funktion  $e : X^2 \rightarrow [0, 1]$  und fordern, dass  $e(a, b)$  genau dann den Wert 1 annimmt, wenn  $a = b$ . Bestimmte Ähnlichkeitsmaße erlauben eine sehr schnelle Auswertung.

**Definition 3.5** Sei  $Y$  eine Menge und sei  $\mathcal{F}$  eine Menge von Funktionen  $h : X \rightarrow Y$ . Dann heißt  $\mathcal{F}$  eine Klasse von ähnlichkeitsbewahrenden Hashfunktionen für  $X$  und das Ähnlichkeitsmaß  $e$ , wenn

$$\text{prob}_{h \in \mathcal{F}}[h(x) = h(y)] = e(x, y).$$

Wenn  $\mathcal{F}$  eine Klasse von ähnlichkeitsbewahrenden Hashfunktionen für  $X$  und  $e$  ist, dann genügt die zufällige Auswahl weniger Hashfunktionen, um die Ähnlichkeit zweier Elemente  $x$  und  $y$  approximativ zu bestimmen.

### Beispiel 3.1 Hamming-Ähnlichkeit

Wir wählen  $X = \{0, 1\}^n$  und  $e(x, y) = 1 - H(x, y)/n$  als Ähnlichkeitsmaß, wobei  $H$  den Hamming-Abstand bezeichnet, also  $H(x, y) = |\{i \mid x_i \neq y_i\}|$ . Dann ist

$$\mathcal{F} = \{p_i \mid p_i(x) = x_i\}$$

eine Klasse ähnlichkeitsbewahrender Hashfunktionen für  $\{0, 1\}^n$  und das Ähnlichkeitsmaß  $e$ , denn  $e(x, y) = 1 - H(x, y)/n = (n - H(x, y))/n = |\{i \mid x_i = y_i\}|/n$  ist die Wahrscheinlichkeit, dass  $x$  und  $y$  in einer zufällig gezogenen Bitposition übereinstimmen.

**Beispiel 3.2 Der Jaccard-Koeffizient und Min-Hashing**

Sei  $X = \mathcal{P}(U)$  die Klasse aller Teilmengen eines endlichen Universums  $U$ . Wir definieren den Jaccard-Koeffizienten  $J(A, B)$  für zwei Teilmengen  $A$  und  $B$  durch

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Beachte, dass  $0 \leq J(A, B) \leq 1$  und  $J(A, B)$  ist genau dann Eins, wenn  $A$  und  $B$  übereinstimmen. Für eine Permutation  $\pi$  von  $U$  setzen wir  $\min_\pi(A) = \min\{j \mid \pi(j) \in A\}$  und  $\min_\pi$  ist der „kleinste Index eines Elements von  $A$ , das unter  $\pi$  in die Menge  $A$  geworfen wird. Schließlich wählen wir  $\mathcal{F} = \{\min_\pi \mid \pi\}$ . Die Klasse  $\mathcal{F}$  definiert das wichtige Verfahren des **Min-Hashing**.

$\mathcal{F}$  ist eine Klasse ähnlichkeitsbewahrender Hashfunktionen für  $X$  und  $J$ . Warum? Für Teilmengen  $A$  und  $B$  von  $U$  sei  $r \in A \cup B$  der kleinste Index eines Elements in  $A \cup B$ , das unter  $\pi$  in die Menge  $A \cup B$  geworfen wird. Es gibt insgesamt  $|A \cup B|$ -viele Möglichkeiten für die Wahl von  $r$ , von denen aber genau  $|A \cap B|$ -viele zur Gleichheit  $\min_\pi(A) = \min_\pi(B)$  führen.

**Beispiel 3.3 Ähnlichkeit von Vektoren**

Diesmal sei  $X = \{x \in \mathbb{R}^n \mid \|x\| = 1\}$  die  $n$ -dimensionale Sphäre. Für zwei Vektoren  $x, y \in X$  sei  $\phi(x, y)$  der Winkel zwischen  $x$  und  $y$ . Dann ist  $e(x, y) = 1 - \phi(x, y)/\pi$  ein Ähnlichkeitsmaß. Für jeden Vektor  $r \in X$  definieren wir die Funktion  $h_r : X \rightarrow \{0, 1\}$  durch

$$h_r(z) = \begin{cases} 1 & \langle z, r \rangle \geq 0 \\ 0 & \text{sonst.} \end{cases}$$

**Aufgabe 25**

Zeige, dass  $\mathcal{F} = \{h_r \mid r \in X\}$  eine Klasse ähnlichkeitsbewahrender Hashfunktionen für  $X$  und  $e$  ist.

Wir erhalten damit auch ein weiteres ähnlichkeitsbewahrendes Hashing-Verfahren für Teilmengen des Universums  $\{1, \dots, n\}$ : Wir repräsentieren eine Menge  $X$  durch ihren Inzidenzvektor<sup>2</sup>  $e_X$ . Als Ähnlichkeit zweier Teilmengen  $A, B \subseteq \{1, \dots, n\}$  können wir dann die Ähnlichkeit ihrer Inzidenzvektoren wählen und der Ähnlichkeitswert von  $A$  und  $B$  ist dann  $1 - \phi(e_A, e_B)/\pi$ . Beachte, dass der Kosinus des Winkels mit dem inneren Produkt der Inzidenzvektoren, nach Normierung auf Länge 1, übereinstimmt und es ist

$$\cos(\phi(e_A, e_B)) = \frac{\langle e_A, e_B \rangle}{\|e_A\| \cdot \|e_B\|} = \frac{|A \cap B|}{\sqrt{|A| \cdot |B|}}.$$

Also ist  $e(A, B) = \frac{|A \cap B|}{\sqrt{|A| \cdot |B|}}$  ebenfalls ein Ähnlichkeitsmaß, dass *indirekt* schnell mit ähnlichkeitsbewahrendem Hashing ausgewertet werden kann: Approximiere den Winkel  $\phi(e_A, e_B)$  mit ähnlichkeitsbewahrendem Hashing und wende dann den Kosinus an.

In den folgenden Abschnitten beschreiben wir Anwendungen und geben Kriterien für die Existenz oder Nicht-Existenz von ähnlichkeitsbewahrenden Hashfunktionen an. Insbesondere werden wir sehen, dass der Wertebereich  $\{0, 1\}$  im gewissen Sinne erzwungen werden kann.

<sup>2</sup>Die *i*te Komponente des Inzidenzvektors  $e_X \in \{0, 1\}^n$  ist 1 genau dann, wenn  $i \in X$ .

### 3.2.1 Min-Hashing

Wir beschreiben zuerst eine Anwendung des Ähnlichkeitsbewahrenden Hashing in der Entdeckung ähnlicher Web-Dokumente. Ungefähr 20% aller Web-Dokumente scheinen Duplikate oder Fast-Duplikate zu sein. Gründe hierfür sind zum Beispiel lokale Kopien beliebter Webseiten, Mirroring und falsch arbeitende Suchmaschinen-Spinnen. Dieses Phänomen ist für Suchmaschinen äußerst ärgerlich, denn einerseits werden kostbare Ressourcen verschwendet und andererseits wird sich ein Anwender nicht durch sehr ähnliche Dokumente wühlen wollen. Unser Ziel ist deshalb die schnelle Bestimmung aller Paare ähnlicher Dokumente in einer großen Dokumentenmenge.

In [BGMZ] wird deshalb ein Verfahren zur Ähnlichkeitsmessung von Dokumenten vorgeschlagen. Dazu wird ein Dokument  $A$  in *Shingles* (wenige aufeinanderfolgende Worte) aufgebrochen und das Dokument wird durch seine Shingle-Menge  $M_A$  repräsentiert<sup>3</sup>. Zur Messung der Ähnlichkeit zweier Dokumente  $A$  und  $B$  wird dann der Jaccard Koeffizient  $J(M_A, M_B)$  benutzt. Hohe Jaccard-Werte belegen eine entsprechend starke syntaktische Ähnlichkeit. Wir können die Bestimmung von  $J(M_A, M_B)$  durch Ähnlichkeitsbewahrendes Hashing wesentlich erleichtern. Dazu fassen wir  $M_A$  als eine Teilmenge des Universums  $U = \{1, \dots, N\}$  auf (in der Anwendung ist  $N \approx 2^{64}$ ) und wählen  $k$  Permutationen  $\pi_1, \dots, \pi_k$  von  $U$  zufällig aus. Wir definieren dann den *Sketch* des Dokuments  $X$  durch

$$\text{sketch}(A) = (\min_{\pi_i}(M_A) \mid 1 \leq i \leq k)$$

und beachten, dass die erwartete Anzahl übereinstimmender Komponenten von  $\text{sketch}(A)$  und  $\text{sketch}(B)$  mit  $k \cdot J(M_A, M_B)$  übereinstimmt.

In [BGMZ] wird von einem Verfahren zum ‘‘syntaktischen Clustering‘‘ auf 30 Millionen Webseiten berichtet. Dieses Verfahren wurde inzwischen von AltaVista übernommen. Ein Fenster von 10 aufeinanderfolgenden Worten definiert ein Shingle, so dass man davon ausgehen kann, dass gemeinsame Shingles ein starker Hinweis für Ähnlichkeit sind. 100 verschiedene lineare Permutationen  $\pi_i(x) \equiv a_i \cdot x + b_i \pmod{N}$  werden zufällig ausgewürfelt und die Dokumente werden somit als Mengen mit 100 Elementen repräsentiert. Entsprechend erwartet man bei 50% Ähnlichkeit zwischen 2 Dokumenten ungefähr 50% Übereinstimmung der sie repräsentierenden Sketches.

Aufgrund der enormen Daten-Dimensionen (150 Gigabyte vor Sketching und immer noch 24 Gigabyte nach Sketching mit  $k = 100$ ) werden Plattenzugriffe ‘‘panisch‘‘ vermieden. In einer ersten Rechenphase werden Sketches für alle Dokumente berechnet. Danach werden die Dokumente  $k$ -fach mit Hashwert  $\min_{\pi_i}$  gehasht und für alle Dokumentenpaare mit gleichem Hashwert wird der Treffer vermerkt. (Hier wird also eine quadratische Laufzeit toleriert, quadratisch in der Anzahl der Dokumente mit gleichem Hashwert. Die Anzahl dieser Dokumente ist aber meist gering, da schon ein einziger gemeinsamer Treffer für nicht-ähnliche Dokumente unwahrscheinlich ist.) Nach dem  $k$ -fachen Hashing untersucht man den Dokumenten-Graphen, dessen Kanten mit der jeweiligen Trefferzahl beschriftet sind und setzt Kanten permanent ein, wenn die entsprechende Trefferzahl einen vorgegebenen Schwellenwert übersteigt. Sodann werden die Zusammenhangskomponenten als Cluster ähnlicher Webseiten aufgefasst.

---

#### Aufgabe 26

In der obigen Anwendung werden nur lineare Permutationen benutzt. Sei  $\mathcal{F}_{\text{linear}}$  die entsprechende Klasse der

---

<sup>3</sup>Besteht ein Dokument  $A$  aus den Worten  $x_1 \cdots x_n$  und besteht ein Shingle aus 10 aufeinanderfolgenden Worten, dann ist  $M_A = \{x_i \cdots x_{i+9} \mid i \leq n - 9\}$  seine Shingle-Menge. (Shingle ist Englisch für Dachziegel.

Hash-Funktionen. Bestimme zwei Mengen  $A$  und  $B$  so, dass

$$\text{prob}_{h \in \mathcal{F}_{\text{linear}}} [ h(A) = h(B) ] \neq J(A, B).$$

„Lineares Hashing“ gibt somit den Jaccard-Koeffizienten nicht exakt wieder, allerdings sind die experimentellen Erfahrungen positiv.

Eine zweite Anwendung des ähnlichkeitsbewahrenden Hashing findet man im Market-Basket Modell. Hier wird der Inhalt von Warenkörben festgehalten, um das Einkaufsverhalten der Kunden untersuchen zu können. Ein wichtiges Ziel ist die Bestimmung korrelierter Waren  $u$  und  $v$ : Die Ware  $u$  tritt möglicherweise nicht häufig auf; wenn sie aber gekauft wird, dann sollte auch Ware  $v$  mit guter Wahrscheinlichkeit im Warenkorb auftreten.

Für jede Ware  $u$  sei  $S_u$  die Menge der Warenkörbe, in denen  $u$  vorkommt. Um korrelierte Waren bestimmen zu können, stellen wir uns deshalb das Ziel, alle Paarmengen  $\{u, v\}$  von Waren  $u$  und  $v$  zu bestimmen für die der Jaccard-Koeffizient  $J(S_u, S_v)$  hinreichend groß ist. Wir wählen wiederum zufällige Permutationen  $\pi_1, \dots, \pi_k$  der Warenkörbe und definieren

$$\text{signatur}(u) = (\min_{\pi_i}(S_u) \mid 1 \leq i \leq k)$$

als die Signatur der Ware  $u$ . Man verwendet wieder lineare Permutationen. Wir können die Signaturen in einem Datendurchlauf berechnen, wenn wir Signaturen zu Anfang auf  $\infty$  setzen und für jeden neuen Warenkorb die Signaturen der enthaltenen Waren adjustieren.

Natürlich möchte man den quadratischen Aufwand im Vergleich *aller* Warenpaare umgehen. Das folgende Verfahren hat sich als erfolgreich herausgestellt:

Angenommen, wir möchten alle Paare mit einem Jaccard-Koeffizienten von mindestens  $t$  bestimmen. Wenn  $J(S_u, S_v) \geq t$ , dann werden  $S_u$  und  $S_v$  in  $r$  fixierten Komponenten der Signatur mit Wahrscheinlichkeit mindestens  $t^r$  übereinstimmen. Wähle deshalb  $r$  größtmöglich, so dass  $\frac{k}{r} \cdot t^r \approx 1$  und wir können einen Treffer erwarten, wenn wir die  $k$  Komponenten in  $\frac{k}{r}$  Gruppen von jeweils  $r$  Komponenten zerlegen und **Gleichheit** auf mindestens einer Gruppe fordern.

Wenn wir jetzt nach  $\min_{\pi_1}, \dots, \min_{\pi_k}$  hashen, dann können wir sehr viele Kandidatenpaare ausschließen; die (hoffentlich) wenigen verbleibenden Kandidatenpaare können dann vollständig durchforstet werden.

Wir lernen eine weitere Anwendung in Abschnitt 4.3.3 kennen.

### 3.2.2 Wann existieren ähnlichkeitsbewahrende Hashfunktionen?

Wir möchten als Nächstes Kriterien entwickeln, um die Existenz von ähnlichkeitsbewahrenden Hashfunktionen nachzuweisen, bzw. ausschließen zu können. Angenommen, das Paar  $(X, e)$  besitzt eine Klasse  $\mathcal{F}$  von ähnlichkeitsbewahrenden Hashfunktionen, besitzt  $(X, e)$  dann auch ähnlichkeitsbewahrendes Hashing mit Wertebereich  $\{0, 1\}$ ?

Die Funktionen in  $\mathcal{F}$  mögen die Menge  $Y$  als Wertebereich besitzen. Dann wählen wir eine Klasse  $\mathcal{G}$  von Hashfunktionen mit Wertebereich  $\{0, 1\}$  und der Eigenschaft, dass

$$\text{prob}_{g \in \mathcal{G}} [ g(y_1) = g(y_2) ] = 1/2, \text{ für alle } y_1, y_2 \in Y \text{ mit } y_1 \neq y_2.$$

Wir betrachten die Menge  $\mathcal{F}_{\mathcal{G}} = \{g \circ f \mid g \in \mathcal{G} \text{ und } f \in \mathcal{F}\}$  von Hashfunktionen. Wenn  $x_1$  und  $x_2$  zwei verschiedene Elemente aus  $X$  sind, dann ist nach Annahme  $\text{prob}_{f \in \mathcal{F}} [ f(x_1) =$

$f(x_2) ] = e(x_1, x_2)$ . Also folgt

$$\text{prob}_{f \in \mathcal{F}, g \in \mathcal{G}}[g(f(x_1)) = g(f(x_2))] = e(x_1, x_2) + \frac{1 - e(x_1, x_2)}{2} = \frac{1 + e(x_1, x_2)}{2}$$

und wir haben das folgende Ergebnis erhalten:

**Lemma 3.6** *Wenn  $(X, e)$  ähnlichkeitsbewahrende Hashfunktionen besitzt, dann besitzt  $(X, \frac{1+e(x_1, x_2)}{2})$  ähnlichkeitsbewahrende Hashfunktionen mit Wertebereich  $\{0, 1\}$ .*

Die Klasse  $\mathcal{F}_{\mathcal{G}}$  möge genau die  $d$  Hashfunktionen  $h_1, \dots, h_d$  besitzen. Dann können wir die Menge  $X$  in den  $d$ -dimensionalen Würfel einbetten, wenn wir die Funktion  $h = (h_1, \dots, h_d) : X \rightarrow \{0, 1\}^d$  benutzen. Die Einbettung  $h$  hat aber noch eine interessante Eigenschaft. Für verschiedene Elemente  $x_1, x_2 \in X$  gilt

$$\text{prob}_i[h_i(x_1) \neq h_i(x_2)] = 1 - \frac{1 + e(x_1, x_2)}{2} = \frac{1 - e(x_1, x_2)}{2}$$

und damit ist der Hamming-Abstand zwischen  $x_1$  und  $x_2$  nach Einbettung in den Würfel genau  $d \cdot \frac{1 - e(x_1, x_2)}{2}$ . Da die Hamming-Distanz eine Metrik ist, muss also auch  $d(x_1, x_2) = \frac{1 - e(x_1, x_2)}{2}$  eine Metrik sein, die sich darüberhinaus in den Würfel einbetten lässt!

**Definition 3.7** Sei  $X$  eine Menge und sei  $d$  eine Metrik auf  $X$ . Wir sagen, dass  $d$  eine Hamming-Metrik ist, wenn es ein  $N$  und eine Funktion  $\Phi : X \rightarrow \{0, 1\}^N$  gibt, so dass

$$H(\Phi(x), \Phi(y)) = \alpha \cdot d(x, y)$$

für eine Konstante  $\alpha$ .

Wir können unsere Ergebnisse jetzt wie folgt zusammenfassen:

**Satz 3.8** *Wenn  $(X, e)$  eine endliche Menge von ähnlichkeitsbewahrenden Hashfunktionen besitzt, dann ist  $1 - e$  eine Hamming-Metrik und insbesondere ist  $1 - e$  eine Metrik. Ist andererseits  $1 - e$  eine Hamming-Metrik, dann gibt es eine Konstante  $\beta$  und  $\frac{\beta + e(x, y)}{\beta + 1}$  besitzt ähnlichkeitsbewahrende Hashfunktionen.*

**Beweis:** Nur die zweite Aussage ist zu verifizieren. Wir wissen gemäß Beispiel 3.1, dass  $1 - H/N$  ähnlichkeitsbewahrendes Hashing besitzt. Nach Annahme ist  $H(\Phi(x), \Phi(y)) = \alpha \cdot (1 - e(x, y))$  und deshalb besitzt

$$\begin{aligned} 1 - \frac{H(\Phi(x), \Phi(y))}{N} &= 1 - \alpha \cdot \frac{(1 - e(x, y))}{N} = \frac{N - \alpha + \alpha \cdot e(x, y)}{N} \\ &= \frac{(N - \alpha)/\alpha + e(x, y)}{(N - \alpha)/\alpha + 1} = \frac{\beta + e(x, y)}{\beta + 1} \end{aligned}$$

ähnlichkeitsbewahrendes Hashing. □

**Beispiel 3.4** Der Überlappungskoeffizient zweier Menge  $A$  und  $B$  ist durch

$$e(A, B) = \frac{|A \cap B|}{\min\{|A|, |B|\}}$$

definiert. Wir zeigen, dass  $e$  kein Ähnlichkeitsbewahrendes Hashing besitzt. Wir definieren die Mengen  $A = \{a\}$ ,  $B = \{b\}$  und  $C = \{a, b\}$  und zeigen, dass die Dreiecksungleichung  $(1-e(A, C)) + (1-e(C, B)) \geq (1-e(A, B))$  verletzt ist. Dies ist offensichtlich, denn  $e(A, B) = 0$  und  $e(A, C) = 1 = e(C, B)$ .

Auch der Dice-Koeffizient

$$e(A, B) = \frac{|A \cap B|}{(|A| + |B|)/2}$$

besitzt kein Ähnlichkeitsbewahrendes Hashing: Wir wählen  $A, B$  und  $C$  wie oben und erhalten  $e(A, B) = 0$  und  $e(A, C) = \frac{2}{3} = e(C, B)$ .

---

#### Aufgabe 27

$X$  sei die Menge aller Teilmengen eines Universums  $U$ . Wir betrachten das Ähnlichkeitsmaß  $e(A, B) = \frac{|A \cap B|}{\sqrt{|A| \cdot |B|}}$ . Zeige, dass  $(X, e)$  kein Ähnlichkeitsbewahrendes Hashfunktionen besitzt. Trotzdem kann  $e$  effizient durch einen randomisierten Algorithmus ausgewertet werden. Warum?

---

### 3.3 Bloom-Filter

Sei  $U$  ein Universum und sei  $\mathcal{H}(U, m)$  die Menge aller Funktionen, die  $U$  auf die Menge  $\{1, \dots, m\}$  abbilden. Ein Bloom-Filter repräsentiert eine Menge  $X \subseteq U$  durch ein Boole'sches Array  $B$  der Länge  $m$  und benutzt dazu  $k$  „Hashfunktionen“  $h_1, \dots, h_k \in \mathcal{H}(U, m)$ :

Anfänglich ist  $X = \emptyset$  und alle Zellen von  $B$  sind auf Null gesetzt. Ein Element  $x \in U$  wird in die Menge  $X$  eingefügt, indem das Array  $B$  nacheinander an den Stellen  $h_1(x), \dots, h_k(x)$  auf Eins gesetzt wird.

Um nachzuprüfen, ob  $x$  ein Element von  $X$  ist, wird  $B$  an den Stellen  $h_1(x), \dots, h_k(x)$  überprüft. Wenn  $B$  an allen Stellen den Wert 1 besitzt, dann wird die Vermutung „ $x \in X$ “ ausgegeben und ansonsten wird die definitive Ausgabe „ $x \notin X$ “ getroffen.

Offensichtlich erhalten wir konventionelles Hashing aus Bloom-Filtern für  $k = 1$ . Wir beachten, dass eine negative Antwort auf eine  $x \in X$ ? Anfrage stets richtig ist. Eine positive Antwort kann allerdings falsch sein und Bloom-Filter produzieren damit „falsche Positive“. Wie groß ist die Wahrscheinlichkeit einer falschen positiven Antwort, wenn eine Menge  $X$  der Größe  $n$  durch ein Boole'sches Array der Größe  $m$  mit Hilfe von  $k$  zufällig aus  $\mathcal{H}(U, m)$  gewählten Hashfunktionen repräsentiert wird?

Angenommen wir haben die Elemente aus  $X$  in das Array  $B$  gehasht. Wie groß ist die Wahrscheinlichkeit  $p_{n,m,k}$ , dass  $B$  an einer zufällig ausgewählte Position eine Null speichert? Offensichtlich folgt mit Lemma 1.2

$$e^{-(kn/m) \cdot (1/(1-1/m))} \leq p_{n,m,k} = \left(\frac{m-1}{m}\right)^{kn} = \left(1 - \frac{1}{m}\right)^{kn} \leq e^{-kn/m},$$

denn es ist  $e^{x/(1+x)} \leq 1 + x \leq e^x$  für  $x > -1$ . Die Abschätzung für  $p_{n,m,k}$  ist scharf, falls  $kn$  und  $m$  die gleiche Größenordnung besitzen und falls  $m$  hinreichend groß ist, denn

$$e^{-(kn/m) \cdot (1/(1-1/m))} = e^{-(kn/m) \cdot (1+1/(m-1))} = e^{-(kn/m)} \cdot e^{-kn/(m \cdot (m-1))} \approx e^{-(kn/m)}.$$

Die Wahrscheinlichkeit  $q_{n,m,k}$  einer falschen positiven Antwort ist beschränkt durch die Wahrscheinlichkeit, in  $k$  Versuchen jeweils eine Eins „zu ziehen“. Wenn wir die Anzahl der Einsen

kennen, dann ist das  $k$ -malige Ziehen von Einsen äquivalent zu  $k$  unabhängigen Experimenten, in denen jeweils geprüft wird, ob eine Eins gezogen wurde. Die Anzahl der Einsen ist aber nicht vorher bekannt und Unabhängigkeit liegt nicht vor wie die folgende Aufgabe zeigt.

---

**Aufgabe 28**

Wir vergleichen die tatsächliche Wahrscheinlichkeit  $q_{n,m,k}$  einer falschen positiven Antwort mit der Abschätzung

$$Q := \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k.$$

Gib ein Beispiel mit  $|U| = 2$ ,  $n = 1$  und  $k = m = 2$  an, für das  $Q < q_{n,m,k}$  gilt. Die tatsächliche Wahrscheinlichkeit einer falschen positiven Antwort kann also größer als die Abschätzung  $Q$  sein!

---

Es stellt sich aber heraus, dass  $Q$  eine sehr scharfe Approximation von  $q_{n,m,k}$  ist. Wir verwenden diese Beziehung ohne weitere Argumentation und erhalten:

$$q_{n,m,k} \approx \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx (1 - e^{-kn/m})^k = e^{k \cdot \ln(1 - e^{-kn/m})}.$$

Wieviele Hash-Funktionen sollten wir bei gegebener Anzahl  $n$  der gespeicherten Elemente und bei gegebener Größe  $m$  des Boole'schen Arrays benutzen, um die Wahrscheinlichkeit einer falschen positiven Antwort zu minimieren?

Einerseits "sollte"  $k$  möglichst groß sein, da dann die Wahrscheinlichkeit wächst eine Null zu erwischen, aber andererseits "sollte"  $k$  möglichst klein sein, da dann eine Einfüge-Operation nur wenige neue Einsen einträgt.

Das optimale  $k$  wird die Wahrscheinlichkeit  $g(k) = q_{n,m,k}$  minimieren und damit auch die Funktion  $\ln g(k)$  minimieren. Wir beachten  $\ln g(k) = -\frac{m}{n} \cdot \ln(p) \cdot \ln(1-p)$  für  $p = e^{-kn/m}$  und erhalten ein globales Minimum für  $p = \frac{1}{2}$ . Die Setzung  $e^{-kn/m} = \frac{1}{2}$  führt dann auf

$$k = \ln(2) \cdot m/n$$

mit der Fehlerwahrscheinlichkeit  $q_{n,m,k} \approx (1-p)^k = 2^{-k}$ .

**Lemma 3.9** *Wenn wir  $k = \ln(2) \cdot m/n$  Hashfunktionen verwenden, um eine Menge der Größe  $n$  durch ein Boole'sches Array der Größe  $m$  zu repräsentieren, dann ist die Wahrscheinlichkeit einer falschen positiven Antwort höchstens  $2^{-k}$ .*

*Die Laufzeit einer Insert- oder Lookup-Operation ist durch  $O(k)$  und die Speicherplatzkomplexität ist durch  $O(n)$  beschränkt.*

Führen wir einen Vergleich mit konventionellem Hashing ( $k = 1$ ) durch: Bei einer Hashtabelle der Größe  $m$  ist  $\frac{n}{m}$  die Wahrscheinlichkeit einer falschen positiven Antwort und eine Fehlerwahrscheinlichkeit von höchstens  $\varepsilon$  führt auf die Ungleichung  $\frac{n}{m} \leq \varepsilon$  und damit auf die Forderung  $m \geq \frac{1}{\varepsilon} \cdot n$ . Bloom-Filter erreichen eine Fehlerwahrscheinlichkeit von höchstens  $\varepsilon$  für  $\ln(2) \cdot m/n = k = \log_2(1/\varepsilon)$  und damit für  $m = \log_2(1/\varepsilon) \cdot \frac{n}{\ln(2)}$ .

Insbesondere erreicht Hashing die Fehlerwahrscheinlichkeit  $\varepsilon = \frac{1}{n}$  mit  $n^2$  Hashzellen, während Bloom-Filter nur  $m = n \cdot \log_2(n) / \ln(2)$  Zellen benötigen und dabei  $k = \log_2 n$  Hashfunktionen einsetzen. Allgemeiner genügen  $k = c \cdot \log_2 n = \log_2(n^c)$  Hashfunktionen mit Hashtabellen der

Größe  $\log_2(n^c) \cdot \frac{n}{\ln(2)}$  für eine Fehlerwahrscheinlichkeit von höchstens  $\frac{1}{n^e}$  und  $m = O(n \log_2 n)$  ist für praktische Anwendungen ausreichend. In Anwendungen wird man deshalb

$$\Omega(n) = m = O(n \cdot \log_2 n) \quad \text{und entsprechend} \quad \Omega(1) = k = O(\log_2 n)$$

wählen. Natürlich sollte der Speicherplatz des Bloom-Filters nicht größer sein als der für die Spezifizierung der Menge hinreichende Speicherplatz  $O(n \cdot \log_2 |U|)$ .

Wir haben bisher die Insert- und Lookup-Operationen für Bloom-Filter implementiert, aber die Implementierung einer Delete-Operation ist schwieriger: Wenn wir das Array  $B$  an den Positionen  $h_1(x), \dots, h_k(x)$  auf Null setzen und wenn  $\{h_1(x), \dots, h_k(x)\} \cap \{h_1(y), \dots, h_k(y)\} \neq \emptyset$ , dann wird auch  $y$  entfernt! Hier hilft ein *zählender Bloom-Filter*: Statt einem Boole'schen Array wird ein Array von Zählern benutzt und Elemente werden durch das Hochsetzen (bzw. Heruntersetzen) der entsprechenden Zähler eingefügt (bzw. entfernt).

---

#### Aufgabe 29

Eine  $n$ -elementige Menge  $S \subseteq U$  sei durch einen zählenden Bloom-Filter  $Z$  mit  $m$  Zählern und  $k$  Hashfunktionen  $h_1, \dots, h_k : U \rightarrow \{1, \dots, m\}$  repräsentiert. Wir nehmen an, dass das Heruntersetzen der entsprechenden Zähler bei der Entfernung eines Elements  $x$  nur dann geschieht, wenn  $Z[h_i(x)] \neq 0$  für alle  $i = 1, \dots, k$  gilt. Sei  $p_+$  (bzw.  $p_-$ ) die Wahrscheinlichkeit einer falschen *positiven* (bzw. *negativen*) Antwort.

(a) Verkleinern oder vergrößern sich die Wahrscheinlichkeiten  $p_+$  und  $p_-$ , wenn wir ein Element  $x \in S$  entfernen?

(b) Angenommen,  $N$  Elemente  $x \in U$  werden aus dem Filter entfernt, wobei wir jetzt auch zulassen, falsche Positive zu entfernen. Sei  $p'_+$  die Wahrscheinlichkeit einer falschen positiven Antwort und  $p'_-$  die Wahrscheinlichkeit einer falschen negativen Antwort in diesem neuen Modell. Zeige, dass dann  $p'_+ \leq p_+$ ,  $p'_- \leq N p_+$  gilt.

---

Die Vereinigung von zwei Mengen  $X_1$  und  $X_2$  kann einfach implementiert werden: Wenn  $X_i$  durch  $B_i$  repräsentiert wird, dann repräsentiert  $B_1 \vee B_2$  die Vereinigung  $X = X_1 \cup X_2$ . Für die Berechnung des Durchschnitts könnten wir ähnlich vorgehen und  $X = X_1 \cap X_2$  durch  $B_1 \wedge B_2$  repräsentieren, aber beachte, dass  $B_1 \wedge B_2$  eine möglicherweise echte "Obermenge" der Bloom-Filter Einfügungen für  $X$  ist und dementsprechend nimmt die Wahrscheinlichkeit falscher Positive zu.

**Bemerkung 3.1** Tatsächlich können wir Bloom-Filter bereits mit zwei Hashfunktionen  $g_1, g_2 : U \rightarrow \{0, \dots, p-1\}$  für eine Primzahl  $p$  implementieren, wenn wir eine geringfügig anwachsende Fehlerwahrscheinlichkeit tolerieren. Dazu setzen wir  $m = k \cdot p$  und definieren die  $k$  Hashfunktionen  $h_1, \dots, h_k$  durch

$$h_i(x) = (i-1) \cdot p + (g_1(x) + (i-1) \cdot g_2(x) \bmod p).$$

Wir beachten, dass  $h_i(x) = h_i(y)$  für alle  $i$  genau dann eintritt, wenn  $g_1(x) = h_1(x) = h_1(y) = g_1(y)$  und  $g_2(x) = g_2(y)$ , und wir haben die Schwäche dieses Ansatzes herausgefunden: Wenn wir eine Menge  $X = \{x_1, \dots, x_n\}$  von  $n$  Elementen aufgebaut haben, dann gilt für  $y \notin X$

$$\text{prob}[\exists j : h_1(y) = h_1(x_j) \wedge \dots \wedge h_k(y) = h_k(x_j)] \leq \frac{n}{p^2}$$

und diese Fehlerwahrscheinlichkeit lässt sich nicht mit wachsendem  $k$  absenken! Aber in praktischen Anwendungen wird man  $p \approx n$  wählen und wir erhalten die relativ kleine Fehlerwahrscheinlichkeit von ungefähr  $1/n$ . Aber  $y$  kann auch falsch positiv sein, wenn es zu jedem  $i$  ein  $j$  mit  $h_i(y) = h_i(x_j)$  gibt und  $y$  mit jedem  $x_j$  höchstens eine gemeinsame Hashzelle besitzt. In diesem Fall kann aber gezeigt werden, dass die Fehlerwahrscheinlichkeit wiederum durch  $(1 - e^{-kn/m})^k$  abgeschätzt werden kann!

Der Vorteil des neuen Schemas ist die schnellere Berechnung der  $k$  Hashfunktionen.

**Aufgabe 30**

Wir zerlegen die  $m$  Positionen von  $B$  in  $k$  Gruppen von jeweils  $m/k$  Positionen und würfeln dann Hashfunktionen  $h_1, \dots, h_k \in \mathcal{H}(U, m/k)$  zufällig aus, wobei  $h_i$  für die  $i$ te Gruppe von Positionen zuständig ist.

Bestimme die Wahrscheinlichkeit  $p$ , dass  $B$  an einer zufällig ausgewählten Position eine Null speichert. Sinkt oder steigt die Wahrscheinlichkeit einer falschen positiven Antwort im Vergleich zum konventionellen Ansatz?

**3.3.1 Anwendungen**

**Verteiltes Caching:** In Web Cache Sharing speichern mehrere Proxies Webseiten in ihren Caches und kooperieren, um zwischengespeicherte Webseiten abzurufen: Bei Nachfrage nach einer bestimmten Seite ermittelt der zuständige Proxy, ob ein Cache eines anderen Proxies die nachgefragte Seite enthält und bittet diesen Proxy um Lieferung. In diesem Szenario liegt die Benutzung von Bloom-Filtern nahe. Jeder Proxy speichert in einem Bloom-Filter ab, welche Seiten zwischengespeichert wurden und sendet in regelmäßigen Abständen seinen aktualisierten Bloom-Filter an seine Kollegen. Natürlich ist es möglich, dass ein Proxy aufgrund des Phänomens falscher Positiver vergebens kontaktiert wird und die Laufzeit erhöht sich in diesem Fall. Allerdings entstehen falsche Positive wie auch falsche Negative bereits durch die Auslagerung von Seiten während eines Aktualisierungszyklus und die Fehler des Bloom-Filters erscheinen durch die Verringerung des Netzwerk-Verkehrs mehr als amortisiert.

**Peer-to-Peer Netzwerke:** Im Synchronisierungsproblem besitzen die Knoten  $X$  eines Netzwerks Mengen  $S_X$  von Objekten. Jeder Knoten  $X$  versucht, alle Objekte zu erfahren, die zu den Mengen seiner Nachbarn, aber nicht zu seiner Menge  $S_X$  gehören. Um dies zu erreichen schickt  $X$  den Bloom-Filter der Menge  $X$  an jeden seiner Nachbarn; Nachbar  $Y$  antwortet mit der unkodierten Menge  $S_Y \setminus S_X$ . Es werden nur tatsächlich benötigte Objekte versandt, aber falsche Positive im Bloom-Filter von  $S_X$  führen dazu, dass einige Objekte in  $S_Y \setminus S_X$  nicht erkannt werden.

**Aggressive Flüsse im Internet Routing:** Die Ermittlung aggressiver Flüsse, also die Ermittlung von Start-Ziel Verbindungen, die Datenstaus durch den Transfer von besonders vielen Paketen verschärfen oder sogar verursachen, ist ein wichtiges Problem in der Internet Verkehrskontrolle.

Um aggressiven Flüssen auf die Schliche zu kommen, repräsentiert ein Router die Menge aller Start-Ziel Paare, deren Pakete er befördert, durch einen zählenden Bloom-Filter. Insbesondere werden für eine Start-Ziel Verbindung  $x$  die Zähler aller Positionen  $h_1(x), \dots, h_k(x)$  inkrementiert. Überschreitet der minimale der  $k$  Werte einen vorgegebenen Schwellenwert  $T$ , dann wird  $x$  als "möglicherweise" aggressiv gebrandmarkt. Beachte, dass alle aggressiven Flüsse erkannt werden; allerdings ist es nicht ausgeschlossen, dass auch unschuldige Flüsse als aggressiv eingeschätzt werden. Wir kommen auf dieses Verfahrens in Abschnitt 4.3.2 zurück.

**Aufgabe 31**

Wir nehmen an, dass eine Menge  $F$  von  $n$  Flüssen in einen Bloom-Filter mit  $m$  Zählern  $Z[i]$ ,  $i = 1, \dots, m$  eingefügt wird. Angenommen,  $v_x$  Pakete eines Start-Ziel Paares  $x \in F$  durchlaufen unseren Router.

(a) Was ist der Zusammenhang zwischen  $\text{prob}[\min\{Z[h_1(x)], \dots, Z[h_k(x)]\} > v_x]$  und der Wahrscheinlichkeit  $p_+$  einer falschen positiven Antwort?

(b) Tatsächlich können wir unseren Ansatz für Anwendungen in der Praxis noch verbessern. Wie sollte eine Modifikation aussehen, die Überschätzungen der Vielfachheiten  $v_x$  für  $x \in F$  erschwert? Insbesondere, wenn  $v'_x$  die Abschätzung unseres Verfahrens ist, und wenn  $v''_x$  die Abschätzung deines neuen Verfahrens ist, dann sollte stets  $v_x \leq v''_x \leq v'_x$  gelten und die Ungleichung  $v''_x < v'_x$  sollte in einigen Fällen auftreten können.

Hier genügt eine kurze Antwort.

**IP-Traceback:** Hier nehmen wir an, dass ein Router durch eine Paketflut angegriffen wird. Unser Ziel ist die Bestimmung aller Wege, über die das Opfer angegriffen wird. In unserer Lösung protokolliert ein Router jedes transportierte Paket in einem Bloom-Filter. Ein angreifendes Paket kann jetzt vom Opfer an alle Nachbarn weitergereicht werden. Der Nachbar, der das Paket befördert hat, wird dies erkennen, wobei aber falsche Positive dazu führen, dass Nachbarn fälschlicherweise annehmen, am Transport beteiligt gewesen zu sein. Wird dieser Rekonstruktionsprozess iteriert, dann sollten diese falschen Verzweigungen aber, bei entsprechend kleiner Wahrscheinlichkeit für falsche Positive, hochwahrscheinlich aussterben. Wir beschreiben eine zweite Methode des IP-Tracebacks in Abschnitt 8.

### 3.4 Zusammenfassung

Wir haben mit DHT's (*distributed hash tables*) als einer fundamentalen Datenstruktur für Peer-to-Peer Systeme begonnen und haben die Implementierung CHORD besprochen. Die wesentliche Idee hinter CHORD ist die Verwendung konsistenter Hashfunktionen  $h : \mathcal{R} \cup \mathcal{D} \rightarrow [0, 1]$  für Rechner aus der Menge  $\mathcal{R}$  aller Rechner und Dokumente aus der Menge  $\mathcal{D}$  aller Dokumente. Die gemeinsame Hashfunktion sichert hochwahrscheinlich eine gleichmäßige Belastung der beteiligten Rechner und eine logarithmische Suchzeit ist selbst dann hochwahrscheinlich, wenn Rechner zwischenzeitlich kommen und gehen. CHORD ist eine zufriedenstellende Lösung für gutartige Peers, aber kryptographische Methoden müssen im Fall bössartiger Peers hinzugezogen werden.

Für die approximative Auswertung bestimmter Ähnlichkeitsmaße haben wir *ähnlichkeitsbewahrendes Hashing* angewandt und dabei vor allem das Ähnlichkeitsmaß von Jaccard betont. Ähnlichkeitsbewahrendes Hashing ist eine fundamentale algorithmische Methode, um alle „hinreichend ähnlichen“ Paare aus einer Menge  $X$  von Dokumenten zu bestimmen: Nach zufälliger Wahl von  $k$  Hashfunktionen  $h_1, \dots, h_k$  wird jedes Dokument  $D \in X$   $k$ -mal gehasht und erhält eine *kurze* Signatur  $(h_1(D), \dots, h_k(D))$ . Ähnliche Dokumente werden dann auch hochwahrscheinlich ähnliche Signaturen besitzen und wir haben die Ähnlichkeit langer Objekte auf die Ähnlichkeit kurzer Objekte reduziert.

*Bloom-Filter* erlauben eine äußerst speicherplatz-effiziente Darstellung von Mengen durch Boole'sche Arrays. Beteiligt sind wiederum  $k$  Hash-Funktionen  $h_1, \dots, h_k$ , die beim Einfügen eines Elements  $x$  das Boole'sche Array an den Stellen  $h_1(x), \dots, h_k(x)$  auf Eins setzen. Während negative Antworten bei einer Suche nach einem Element stets richtig sind, können positive Antworten falsch sein. Wir haben festgestellt, dass die Wahrscheinlichkeit falscher positiver Antworten bei  $n$  gespeicherten Elementen und einem Boole'schen Array der Länge  $m$  höchstens  $2^{-k}$  beträgt, wenn  $k = \ln(2) \cdot m/n$  gewählt wird.



# Kapitel 4

## Streaming Data

Abhängig von der Größe der Datenmengen werden verschiedene Speichermodelle benutzt:

- Im *Hauptspeichermodell* wird angenommen, dass der Hauptspeicher für die Berechnung ausreichend ist.
- Im *Sekundärspeichermodell* sind die Daten zum Beispiel auf der Platte zu speichern. Jetzt ist die Suchzeit (von ungefähr 4 ms) die teure Ressource und dementsprechend ist die Anzahl der Plattenzugriffe zu minimieren.
- Das *Streaming-Data Modell* beschreibt ein On-line Szenario: Daten strömen fortlaufend ein und Berechnungen sind in Echtzeit, bzw in wenigen Datendurchläufen zu erbringen. Beispiele sind Verkehrsmessungen im Internet, die Datenanalyse in der Abwehr einer Denial-of-Service Attacke, die Verarbeitung von durch Satelliten erfassten Daten oder die fortlaufende Protokollierung von Telefonverbindungen durch weltweit agierende Telefonunternehmen und die damit verbundenen Reaktionen auf überlastete Leitungen.

Im Streaming Data Modell erhalten wir einen Datenstrom  $(x_i | i)$ , wobei jedes  $x_i$  zu dem Universum  $U = \{1, \dots, m\}$  der Größe  $m$  gehört. Zur Orientierung sollte man  $m$  im Bereich von mehreren Megabyte bis hin zum Gigabyte-Bereich annehmen. Damit sind im Streaming Data Modell sehr hohe Anforderungen vorgegeben: Gewaltige Datenmengen sind schnellstmöglich zu bewältigen, wobei im Regelfall nur ein Datendurchlauf zur Verfügung steht. Wir werden uns nun mit den folgenden algorithmischen Fragestellungen beschäftigen:

**Stichproben** reduzieren die Größe der Datenmenge und sind deshalb ein wichtiges Hilfsmittel für viele einfache Probleme. Beachte, dass eine Stichproben nicht ein und für alle Mal bestimmt werden kann, sondern sich mit dem Datenstrom dynamisch ändern muss.

**Häufigkeitsanalyse für Datenströme:**  $a_u = |\{i | x_i = u\}|$  ist die Häufigkeit des Schlüssels  $u$ . Wir beschäftigen uns mit der exakten oder approximativen Bestimmung des  $k$ ten Häufigkeitsmoments

$$H_k = \sum_{u \in U} a_u^k.$$

$H_0$  ist die Anzahl der verschiedenen Schlüssel, denn es ist  $x^0 = 1$  für  $x \neq 0$  und  $x^0 = 0$  sonst. Das zweite Häufigkeitsmoment  $H_2$  misst wie gleichmäßig die Datenmenge auf die einzelnen Schlüssel verteilt ist: Es ist  $m \cdot \left(\frac{n}{m}\right)^2 = \frac{n^2}{m} \leq H_2 \leq n^2$  und kleine Werte von  $H_2$  implizieren eine gleichmäßige Verteilung.

Wir sind ebenfalls an der Bestimmung seltener und häufiger Schlüssel interessiert.

**Zeitfenster:** Um das Verhalten eines Datenstroms in der jüngsten Vergangenheit bestimmen zu können, wertet man Datenströme in *Zeitfenstern* aus. Bei entsprechend großen Zeitfenstern ist die Abspeicherung des Datenstroms nicht möglich und wir müssen Methoden entwickeln, um die quantitative Analyse nach dem Verschwinden veralteter und dem Erscheinen junger Schlüssel zu aktualisieren.

**Bestimmung von Histogrammen:** Ein Histogramm komprimiert den Datenstrom und bestimmt die charakteristischen numerischen Werte innerhalb weniger Zeitintervalle.

---

**Aufgabe 32**

- (a) Die Zahlen  $1, \dots, n$  werden in einer vorher nicht bekannten Permutation ausgegeben. Allerdings fehlt genau eine der Zahlen. Bestimme die fehlende Zahl mit Speicherplatz  $O(\log_2 n)$ .
- (b) Wie in Teil (a) werden die Zahlen  $1, \dots, n$  wieder permutiert, allerdings fehlen diesmal genau  $k$  Zahlen. Bestimme die fehlenden Zahlen mit Speicherplatz  $O(\log_2 n)$ .
- 

## 4.1 Stichproben

Unser Ziel ist die Auswahl einer relativ kleinen Stichprobe aus einem Datenstrom  $(x_i \mid i)$ , und wir stellen die Berechnung einer ohne Ersetzung gleichverteilt gezogenen Stichprobe  $S \subseteq \{(i, x_i) \mid 1 \leq i \leq n\}$  vor. Natürlich ist die Hoffnung, dass statistische Anfragen direkt auf der sehr viel kleineren Stichprobe beantwortet werden können.

### Algorithmus 4.1 Reservoir Sampling

- (1)  $T$  sei eine obere Schranke für die Größe der Stichprobe. Der Parameter  $t$  zählt die bisher gesehenen Schlüssel; setze  $t = 0$  und STICHPROBE =  $\emptyset$ .
- (2) Durchlaufe die Schlüssel nacheinander:
  - (2a) Setze  $t = t + 1$ .
  - (2b) Wenn  $t \leq T$ , dann füge den Schlüssel in STICHPROBE ein.
  - (2c) Wenn  $t > T$ , dann werfe eine Münze mit Erfolgswahrscheinlichkeit  $\frac{T}{t}$ . Bei einem Erfolg bestimme zufällig einen Schlüssel aus STICHPROBE und entferne den Schlüssel; der aktuelle Schlüssel wird eingefügt. Bei einem Misserfolg wird nichts unternommen.

*Kommentar:* Die Stichprobengröße  $T$  bleibt unverändert.

**Satz 4.2** *Zu jedem Zeitpunkt  $t \geq T$  wird jede  $T$ -elementige Teilmenge  $X \subseteq \{(i, x_i) \mid 1 \leq i \leq t\}$  mit Wahrscheinlichkeit  $1/\binom{t}{T}$  als Stichprobe gewählt.*

**Beweis:** Wir führen eine Induktion nach  $t$ . Die Aussage ist offenbar für  $t = T$  richtig. Wir nehmen an, dass die Aussage für  $t - 1$  richtig ist. Der Schlüssel  $(t, x_t)$  wird dann mit Wahrscheinlichkeit  $\frac{T}{t}$  aufgenommen und das ist genau die Wahrscheinlichkeit, dass eine zufällige  $T$ -elementige Teilmenge  $X \subseteq \{(i, x_i) \mid 1 \leq i \leq t\}$  den Schlüssel  $(t, x_t)$  enthält, denn

$$\binom{t-1}{T-1} / \binom{t}{T} = \frac{T}{t}.$$

Wir können also eine  $T$ -elementige Stichprobe nach dem folgenden Verfahren ziehen:

- Entscheide zuerst mit Erfolgswahrscheinlichkeit  $\frac{T}{t}$ , ob  $(t, x_t)$  gewählt wird.
- Wenn  $(t, x_t)$  nicht gewählt wird, dann wähle eine zufällige,  $T$ -elementige Stichprobe aus der Menge  $\{(i, x_i) \mid 1 \leq i \leq t-1\}$ .
- Wenn  $(t, x_t)$  gewählt wird, dann wähle eine zufällige Stichprobe mit  $T-1$  Elementen aus der Menge  $\{(i, x_i) \mid 1 \leq i \leq t-1\}$ .

Genau dieses Verfahren führt Algorithmus 4.1 durch, da er zuerst mit Wahrscheinlichkeit  $\frac{T}{t}$  entscheidet, ob  $(t, x_t)$  aufzunehmen ist. Wird  $(t, x_t)$  nicht aufgenommen, dann wenden wir die Induktionsvoraussetzung an und erhalten, dass eine zufällige  $T$ -elementige Stichprobe aus  $\{(i, x_i) \mid 1 \leq i \leq t-1\}$  gezogen wurde.

Ansonsten wird  $(t, x_t)$  aufgenommen und wir betrachten die alte Stichprobe  $X$ , von der wir induktiv annehmen können, dass sie einer zufälligen  $T$ -elementigen Teilmenge von  $\{(i, x_i) \mid 1 \leq i \leq t-1\}$  entspricht. Algorithmus 4.1 entfernt ein zufälliges Element aus  $X$  und erhält damit eine zufällige Stichprobe aus  $\{(i, x_i) \mid 1 \leq i \leq t-1\}$  mit  $T-1$  Elementen. Diese modifizierte Stichprobe wird um  $(t, x_t)$  vergrößert und wir erhalten eine zufällige  $T$ -elementige Stichprobe  $X$ .  $\square$

Wir können also das mächtige Hilfsmittel der Berechnung auf Stichproben im Streaming Data Modell benutzen! Als eine erste Anwendung betrachten wir die Bestimmung eines approximativen Medians in einem Datendurchlauf.

#### Algorithmus 4.3 Bestimmung eines approximativen Medians.

- (1) Benutze Reservoir Sampling, um eine Stichprobe  $S$  der Größe  $s$  zu ziehen.
- (2) Bestimme den Median  $M$  von  $S$  und gib  $M$  als Approximation des tatsächlichen Medians aus.

**Satz 4.4**  $\delta, \varepsilon \in [0, 1]$  seien vorgegeben. Wenn Algorithmus 4.3 mit einer Stichprobe der Größe  $s = c \cdot \frac{1}{\varepsilon^2} \cdot \ln \frac{1}{\delta}$  für ein hinreichend großes  $c$  arbeitet, dann liegt der Rang des ausgegebenen Schlüssels mit Wahrscheinlichkeit mindestens  $1 - \delta$  in dem Intervall  $[\frac{n}{2} - \varepsilon \cdot n, \frac{n}{2} + \varepsilon \cdot n]$ .

**Beweis:** Sei  $x_{\text{unten}}$  (bzw.  $x_{\text{oben}}$ ) der Schlüssel des Datenstroms vom Rang  $(\frac{1}{2} - \varepsilon) \cdot n$  (bzw. vom Rang  $(\frac{1}{2} + \varepsilon) \cdot n$ ). Wir haben nur dann Pech gehabt, wenn 50% aller Schlüssel der Stichprobe unterhalb von  $x_{\text{unten}}$  (bzw. oberhalb von  $x_{\text{oben}}$ ) liegen.

Ein Schlüssel kleiner als  $x_{\text{unten}}$  wird mit Wahrscheinlichkeit  $(\frac{1}{2} - \varepsilon)$  gezogen und die erwartete Anzahl dieser „kleinen“ Schlüssel ist deshalb höchstens  $(\frac{1}{2} - \varepsilon) \cdot s$ . Wir haben also Pech, wenn sogar  $\frac{1}{2} \cdot s = (\frac{1}{2} - \varepsilon) \cdot s \cdot (1 + \frac{\varepsilon}{1/2 - \varepsilon})$  kleine Schlüssel gezogen werden. Wir wenden die Chernoff-Schranke an und erhalten die Wahrscheinlichkeit  $e^{-\Omega(\varepsilon^2 \cdot s)}$  als obere Schranke. Um die Fehlerwahrscheinlichkeit auf höchstens  $\delta$  zu beschränken, ist also  $\varepsilon^2 \cdot s = \Omega(\ln \frac{1}{\delta})$  zu fordern. Die Behauptung folgt, da die Anzahl der „großen“ Schlüssel ein analoges Verhalten zeigt.  $\square$

Die zweite Anwendung betrifft ein Clustering Problem, nämlich das  $k$ -Zentren Problem. Hier ist ein vollständiger ungerichteter Graph  $G = (V, E)$  und eine Metrik<sup>1</sup>  $d$  gegeben. Für ein fixiertes  $k$  ist eine Menge  $Z \subseteq V$  von  $k$  Knoten zu bestimmen, so dass

$$\max_{v \in V} \min_{w \in Z} d(v, w)$$

<sup>1</sup> $d : V^2 \rightarrow \mathbb{R}_{\geq 0}$  ist eine Metrik, falls  $d(u, v) = d(v, u)$ ,  $d(u, v) = 0$  gdw.  $u = v$  und  $d(u, v) + d(v, w) \geq d(u, w)$ .

kleinstmöglich ist. Hier wird also nach dem größten Abstand eines Punktes vom nächstliegenden Cluster-Zentrum gefragt. Die Sprachenversion des  $k$ -Zentren Problems ist  $NP$ -vollständig und deshalb wird uns nur eine approximative Lösung gelingen.

#### Algorithmus 4.5 Clustering auf einer Stichprobe

- (1) Die Zahl  $k$  der erlaubten Cluster-Zentren ist gegeben ebenso wie die Metrik  $d$ . Die Folge  $(x_j \mid j)$  bezeichne den Datenstrom.
- (2) Benutze Reservoir Sampling, um eine Stichprobe  $S$  der Größe  $s$  zu ziehen.
- (3) Setze  $Z = \{x_j\}$  für einen beliebigen Schlüssel  $x_j \in S$ . Wiederhole  $k - 1$  mal:
 

Bestimme einen Schlüssel  $x_i \in S$ , dessen minimaler Abstand zu einem Schlüssel in  $Z$  größtmöglich ist. Füge  $x_i$  in die Menge  $Z$  ein.
- (4)  $Z$  wird als Menge der Cluster-Zentren ausgegeben.

Wir zeigen zuerst, dass unsere Wahl für  $Z$  auf der Stichprobe 2-approximativ ist. Dann müssen wir  $Z$  auf dem Datenstrom auswerten.

Angenommen die optimale Cluster-Lösung auf dem Datenstrom besitzt den Radius<sup>2</sup>  $\text{opt}$ . Wenn es einen Punkt  $p \in S$  gibt, der einen Abstand von größer als  $2 \cdot \text{opt}$  zu allen Punkten in  $Z$  besitzt, dann haben gemäß Konstruktion von  $Z$  je zwei Punkte in  $Z \cup \{p\}$  einen Abstand von größer als  $2 \cdot \text{opt}$ . Also gehören die Punkte aus  $Z \cup \{p\}$  zu verschiedenen Clustern der optimalen Lösung: Wenn ein Clusterpunkt  $y$  der optimalen Lösung nächstliegender Punkt für zwei Elemente  $u, v \in Z \cup \{p\}$  ist, dann ist  $d(u, v) \leq d(u, y) + d(y, v) \leq 2 \cdot \text{opt}$ . Aber die optimale Lösung besitzt nur  $k$  Cluster und wir haben einen Widerspruch erhalten: Schritt (3) berechnet eine Menge  $Z$  von Zentren mit  $\max_{v \in S} \min_{w \in Z} d(v, w) \leq 2 \cdot \text{opt}$ .

Wir behaupten, dass Algorithmus 4.5 dieselbe Approximationsleistung auf dem vollständigen Datenstrom erzielt, wenn wir eine kleine Menge von Punkten ausschließen dürfen.

**Satz 4.6** *Ein Datenstrom der Länge  $n$  sei gegeben. Wenn wir eine Stichprobe der Größe  $s = \frac{k \cdot \ln n + \ln(\frac{1}{\delta})}{\varepsilon}$  wählen, dann ist Algorithmus 4.5 mit Wahrscheinlichkeit mindestens  $1 - \delta$  auf einer Teilmenge der Größe  $(1 - \varepsilon) \cdot n$  2-approximativ.*

**Beweis:** Sei  $\text{opt}$  der optimale Radius einer optimalen Cluster-Lösung. Sei  $Z$  eine beliebige, aber „schlechte“ Zentrenmenge: Die Menge  $\text{Weitweg}(Z) = \{x_i \mid d(x_i, Z) > 2 \cdot \text{opt}\}$ , also die Menge aller Punkte mit einem Abstand von mehr als  $2 \cdot \text{opt}$  von ihrem nächstliegenden Zentrum in  $Z$ , habe mehr als  $\varepsilon \cdot n$  Elemente.

Wir zeigen zuerst, dass die Stichprobe  $S$  nur mit einer Wahrscheinlichkeit von höchstens  $\delta / \binom{n}{k}$  die Menge  $\text{Weitweg}(Z)$  nicht trifft.

$$\begin{aligned} \text{prob}[S \cap \text{Weitweg}(Z) = \emptyset] &\leq \left(\frac{(1 - \varepsilon)n}{n}\right)^s = (1 - \varepsilon)^s \\ &\leq e^{-\varepsilon \cdot s} = e^{-(k \cdot \ln n + \ln(\frac{1}{\delta}))} = \delta \cdot n^{-k} \\ &\leq \delta / \binom{n}{k}. \end{aligned}$$

<sup>2</sup>Der Radius ist der längste Abstand eines Datenpunkts von einem Zentrum.

Wenn  $\text{Weitweg}(Z)$  getroffen wird, dann kann Algorithmus 4.5 die schlechte Zentrenmenge  $Z$  nicht gewählt haben, denn sonst hätten wir einen Punkt in  $S$  gefunden, der einen Abstand von mehr als  $2 \cdot \text{opt}$  von  $Z$  besitzt und das haben wir oben ausgeschlossen. Die Wahrscheinlichkeit irgendeine schlechte Zentrenmenge zu wählen ist somit höchstens  $\binom{n}{k} \cdot \delta / \binom{n}{k} = \delta$ . Also wird Algorithmus 4.5 mit Wahrscheinlichkeit mindestens  $1 - \delta$  eine gute Zentrenmenge  $Z$ , also eine Menge  $Z$  mit  $|\text{Weitweg}(Z)| \leq \varepsilon \cdot n$ , bestimmen und das war zu zeigen.  $\square$

---

**Aufgabe 33**

Wir zeigen, dass der Stichprobenansatz eine 2-approximative Lösung nur nach Ausschluss von Punkten erreicht. Insbesondere sei  $s_n = o(n)$ ,  $\delta$  eine Konstante mit  $0 < \delta < 1$  und  $k$  eine natürliche Zahl.

Konstruiere eine Instanz  $(I_n)$  für das  $k$ -Zentren Problem, die aus  $n$  Punkten der Ebene  $\mathbb{R}^2$  besteht. Weise nach, dass für genügend großes  $n$  Algorithmus 4.5, angewandt auf eine Stichprobe der Größe  $s_n$ , mit Wahrscheinlichkeit mindestens  $\delta$  keine 2-approximative Lösung für  $I_n$  liefert.

---

Allerdings ist der Stichprobenansatz selbst für relativ einfache Probleme kein Allheilmittel, denn wir zeigen jetzt, dass der Stichprobenansatz bereits für die approximative Bestimmung der Anzahl verschiedener Schlüssel scheitert. Unser negatives Resultat gilt für eine sehr große Klasse von (deterministischen oder probabilistischen) Algorithmen zur Stichproben-Erstellung, nämlich für alle Algorithmen, die nur  $r \ll n$  Schlüssel inspizieren. Beachte, dass Algorithmus 4.1 nur die in der Stichprobe gesammelten Schlüssel inspizieren kann, denn die Werte der ausgelassenen Schlüssel bleiben unbekannt. Damit wird unser negatives Ergebnis auch für Algorithmus 4.1 gelten. Insbesondere zeigen wir, dass es schwierig ist, die beiden folgenden Szenarien voneinander zu unterscheiden.

**Szenario 1** besteht nur aus den nur mit Einsen besetzten Folge.

**Szenario 2** besteht aus allen Folgen, für die jedes  $i \in \{2, \dots, k\}$  genau einmal auftritt. Alle restlichen Folgeelemente haben den Wert 1.

Unser Ziel ist der Nachweis, dass Stichproben, die nur den Schlüsselwert 1 besitzen, eine relativ hohe Wahrscheinlichkeit haben. Dazu nehmen wir an, dass irgendein Algorithmus  $r < n$  Schlüssel (möglicherweise sogar nur deterministisch) inspiziert.

**Lemma 4.7** Sei  $A$  ein Algorithmus und sei  $X_i$  die Zufallsvariable, die den  $i$ -ten von  $A$  ausgewählten Schlüssel als Wert besitzt. Dann gilt

$$\text{prob}[X_i = 1 \mid X_1 = X_2 = \dots = X_{i-1} = 1] = \frac{n - i - k + 1}{n - i + 1},$$

wenn nur Eingaben des Szenarios 2 auftreten.

**Beweis:** Wir müssen Szenario 2 betrachten und dort den Fall, dass die ersten  $i-1$  inspizierten Schlüssel sämtlich den Wert 1 besitzen. Von den  $n-k$  Schlüsseln mit Wert 1 verbleiben somit  $n-k-(i-1)$  noch nicht inspizierte Schlüssel mit Wert 1. Da insgesamt  $n-(i-1)$  Schlüssel noch nicht inspiziert wurden, folgt die Behauptung.  $\square$

Sei  $I$  das Ereignis, dass alle  $r$  inspizierten Schlüssel den Wert 1 besitzen. Wie wahrscheinlich ist  $I$ ?

$$\begin{aligned} \text{prob}[I] &= \prod_{i=1}^r \text{prob}[X_i = 1 \mid X_1 = X_2 = \dots = X_{i-1} = 1] \\ &= \prod_{i=1}^r \frac{n - i - k + 1}{n - i + 1} \end{aligned}$$

$$\begin{aligned}
&\geq \left(\frac{n-r-k}{n-r}\right)^r \\
&= \left(1 - \frac{k}{n-r}\right)^r \geq e^{-\frac{2 \cdot k \cdot r}{n-r}} \quad \text{falls } \frac{k}{n-r} \leq \frac{1}{2},
\end{aligned}$$

denn wir haben in der letzten Ungleichung benutzt, dass  $1 - z \geq e^{-2 \cdot z}$  für  $0 \leq z \leq \frac{1}{2}$  gilt. Wir setzen  $k = \frac{n-r}{2 \cdot r} \cdot \ln(2)$  und erhalten  $\frac{k}{n-r} \leq \frac{1}{2}$  für  $r \geq 2$  und damit  $\text{prob}[I] \geq 1/2$ .

**Satz 4.8** *Es gelte  $k = \frac{n-r}{2 \cdot r} \cdot \ln(2)$  sowie  $r \geq 2$ . Sei  $A$  ein Algorithmus, der nur  $r$  Schlüssel inspiziert.*

- (a)  *$A$  wird im Szenario 2 mit Wahrscheinlichkeit mindestens  $1/2$  eine Stichprobe bestimmen, die nur aus dem Wert 1 besteht.*
- (b)  *$A$  wird die tatsächliche Anzahl verschiedener Schlüssel mit Wahrscheinlichkeit mindestens  $1/2$  im Szenario 1 oder im Szenario 2 um den Faktor*

$$\sqrt{k} = \sqrt{\frac{n-r}{2 \cdot r} \cdot \ln(2)}$$

*über- bzw. unterschätzen.*

**Beweis:** (a) haben wir bereits gezeigt. (b) folgt direkt aus (a), wenn wir beachten, dass das Szenario 1 genau einen Schlüssel und Szenario 2 genau  $k + 1$  Schlüssel besitzt. Nach Teil (a) kann Algorithmus  $A$  aber mit Wahrscheinlichkeit  $e^{-1}$  nicht zwischen den beiden Szenarien unterscheiden und deshalb wird ein Szenario um den Faktor mindestens  $\sqrt{k+1} \geq \sqrt{k}$  falsch eingeschätzt.  $\square$

## 4.2 Häufigkeitsmomente

Wir beginnen mit negativen Ergebnissen und zeigen, dass die Kommunikationskomplexität für die Berechnung der Häufigkeitsmomente gute untere Schranken für die Speicherplatzkomplexität im Streaming-Data Modell liefert.

### 4.2.1 Streaming-Data und Kommunikation

Wir stellen zuerst das 2-Parteien Kommunikationsmodell vor. Zwei Parteien, Alice und Bob, besitzen Eingaben  $x$  bzw.  $y$ , wobei weder Alice noch Bob die Eingabe des Partners kennen. Im Einweg-Modus schickt Alice eine Nachricht  $\text{message}(x)$  zu Bob und Bob muss das Ergebnis nur in Abhängigkeit von seiner Eingabe  $y$  und der von Alice geschickten Nachricht berechnen. Da wir an dieser Stelle vor Allem an negativen Ergebnissen interessiert sind, nehmen wir an, dass Alice und Bob, im Rahmen der ihnen zur Verfügung stehenden Informationen, eine unbeschränkte Rechenkraft besitzen: Negative Aussagen über das 2-Parteien Kommunikationsmodell sind damit umso stärker.

Deterministische oder probabilistische Protokolle bestimmen die von Alice geschickte Nachricht und definieren die von Bob zu berechnende Antwort. Das Ziel ist die (zumindest approximative) Berechnung einer Funktion  $f(x, y)$ , wobei die Länge der längsten von Alice geschickten Nachricht zu minimieren ist.

**Definition 4.9** Seien  $A$  und  $B$  Mengen von Eingaben und sei  $f : A \times B \rightarrow \mathbb{R}$  mit relativem Fehler höchstens  $\delta$  zu berechnen.

- (a) In einem deterministischen Protokoll erhält Alice eine Eingabe  $x \in A$  und Bob eine Eingabe  $y \in B$ . Das deterministische Protokoll heißt genau dann  $\delta$ -approximativ, wenn Bob für jedes Eingabepaar  $(x, y)$  ein Ergebnis  $a(x, y)$  mit  $(1 - \delta) \cdot f(x, y) \leq a(x, y) \leq (1 + \delta) \cdot f(x, y)$  berechnet.
- (b) Ein probabilistisches Protokoll heißt genau dann  $\delta$ -approximativ mit Fehler  $\varepsilon$ , wenn Bob für jedes Eingabepaar  $(x, y)$  mit Wahrscheinlichkeit mindestens  $1 - \varepsilon$  ein Ergebnis  $a(x, y)$  mit  $(1 - \delta) \cdot f(x, y) \leq a(x, y) \leq (1 + \delta) \cdot f(x, y)$  berechnet.
- (c) Wir definieren die deterministische Kommunikationskomplexität einer  $\delta$ -approximativen Berechnung von  $f$  durch

$$C^\delta(f) = \text{die Länge der längsten Nachricht eines besten } \delta\text{-approximativen deterministischen Protokolls für } f.$$

- (d) Wir definieren die probabilistische Kommunikationskomplexität einer  $\delta$ -approximativen Berechnung von  $f$  durch

$$C_\varepsilon^\delta(f) = \text{die Länge der längsten Nachricht eines besten } \delta\text{-approximativen probabilistischen Protokolls, das } f \text{ mit Fehler } \varepsilon \text{ berechnet.}$$

#### Aufgabe 34

Im Identitätsproblem ist die Funktion  $\text{id}_n : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}$  mit  $\text{id}_n(x, y) = \begin{cases} 1 & x = y \\ 0 & \text{sonst} \end{cases}$  zu

berechnen. Zeige, dass  $C^\delta(\text{id}_n) = n$  gilt, falls  $\delta < \frac{1}{2}$ . (Hinweis: Was passiert, wenn dieselbe Nachricht für zwei verschiedene Eingaben von Alice verschickt werden?)

Probabilistische Protokolle sind sehr viel effizienter, denn es gilt  $C_{1/3}^0(\text{id}_n) = O(\log_2 n)$ . (Hinweis: Interpretiere die beiden Strings  $x$  und  $y$  als Zahlen. Was passiert, wenn wir Gleichheitstests modulo kleiner Primzahlen durchführen?)

Wenn eine Funktion  $f : \{0, 1\}^* \rightarrow \mathbb{R}$  im Streaming-Data Modell zu berechnen ist, dann definieren wir  $f_n$  als die Einschränkung von  $f$  auf Eingaben der Länge  $n$ . Für jede Eingabe  $x$  erhält Alice den Präfix und Bob den Suffix von  $x$  als Eingabe der jeweiligen Länge  $\frac{n}{2}$ .

Wir werden sehen, dass die Kommunikationskomplexität im Allgemeinen gute untere Schranken für die Speicherplatzkomplexität im Streaming-Data Modell ergibt.

**Lemma 4.10** Die Funktion  $f_n$  sei zu berechnen.

- (a) Jeder deterministische Algorithmus, der  $f$   $\delta$ -approximativ mit  $b$  Befehlen berechnet, benötigt mindestens die Speicherkomplexität  $C^\delta(f_n) - O(\lceil \log_2 b \rceil)$ .
- (b) Jeder probabilistische Algorithmus, der  $f$   $\delta$ -approximativ mit Fehler  $\varepsilon$  und  $b$  Befehlen berechnet, benötigt mindestens die Speicherkomplexität  $C_\varepsilon^\delta(f_n) - O(\lceil \log_2 b \rceil)$ .

**Beweis:** Da die Beweise für (a) und (b) fast identisch sind, zeigen wir nur Teil (b). Sei  $A$  ein probabilistischer Algorithmus, der  $f$  im Streaming-Data Modell  $\delta$ -approximativ mit Fehler  $\varepsilon$  berechnet.  $A$  möge die Speicherkomplexität höchstens  $s(n)$  für Eingaben der Länge  $n$  besitzen.

Es genügt, wenn wir ein probabilistisches Protokoll für  $f_n$  entwerfen, das Nachrichten der Länge höchstens  $s(n) + O(1)$  verschickt.

Das Protokoll simuliert Algorithmus  $A$  auf Eingabefolgen der Länge  $n$ , wobei wir annehmen, dass Alice die ersten  $n/2$  Bits und Bob die letzten  $n/2$  Bits erhält. Alice bearbeitet ihre Eingabe mit Algorithmus  $A$  und verschickt dann die entstehende Konfiguration (Speicherinhalt und nächster auszuführender Befehl) an Bob. Bob kann damit die Berechnung von  $A$  problemlos fortsetzen. Da der Algorithmus die Funktion  $\delta$ -approximativ mit Fehler höchstens  $\varepsilon$  berechnet, hat das simulierende Protokoll dieselbe Eigenschaft.

Wenn Algorithmus  $A$  also  $b$  Befehle besitzt, dann folgt  $s(n) + O(\lceil \log_2 b \rceil) \geq C_\varepsilon^\delta(f_n)$  und die Behauptung ist gezeigt.  $\square$

---

**Aufgabe 35**

Warum können wir in Satz 4.10 nicht  $s(n) + \lceil \log_2 b \rceil \geq C_\varepsilon^\delta(f_n)$  folgern?

---

**Fakt 4.1** *Im Disjunktheitsproblem der Größe  $m$  erhalten Alice und Bob Inzidenzvektoren der Teilmengen  $x, y \subseteq \{1, \dots, m\}$ . Die Funktion  $D_m$  sei durch*

$$D_m(x, y) = \begin{cases} 1 & x \cap y = \emptyset \\ 0 & \text{sonst} \end{cases}$$

*definiert. Dann gilt  $C_\varepsilon^\delta(D_m) = \Omega(m)$  für jedes  $\varepsilon < \frac{1}{2}$  und jedes  $\delta < \frac{1}{2}$ .*

Leider erhalten wir als eine erste Konsequenz eine hohe Speicherkomplexität, wenn die größte Häufigkeit eines Schlüssels zu berechnen ist.

**Satz 4.11** *Seien  $\varepsilon, \delta < \frac{1}{2}$  beliebig und sei  $A$  ein probabilistischer Algorithmus, der die größte Häufigkeit  $\delta$ -approximativ mit Fehler höchstens  $\varepsilon$  im Streaming-Data Modell berechnet. Dann benötigt  $A$  Speicherkomplexität  $\Omega(m)$ , wobei  $m$  die Anzahl verschiedener Schlüssel ist.*

**Beweis:** Sei  $A$  ein probabilistischer Algorithmus, der die Häufigkeit des häufigsten Schlüssels approximativ berechnet. Wir zeigen, dass  $A$  zur Lösung des Disjunktheitsproblems benutzt werden kann und weisen dazu Alice und Bob die Teilmengen  $x, y \subseteq \{1, \dots, m\}$  zu. Wir beobachten, dass offensichtlich

$$D_m(x, y) = \begin{cases} 1 & 1 = \text{größte Häufigkeit für } (x, y), \\ 0 & 2 \leq \text{größte Häufigkeit für } (x, y) \end{cases}$$

für die Funktion  $D_m$  des Disjunktheitsproblems gilt und  $A$  liefert wie versprochen auch eine Lösung des Disjunktheitsproblems. Nach Fakt 4.1 ist aber  $C_\varepsilon^\delta(D_m) = \Omega(m)$  und die Behauptung folgt mit Lemma 4.10.  $\square$

**Bemerkung 4.1** Damit nützt weder der Einsatz von probabilistischen Algorithmen noch stellt eine approximative Berechnung eine wesentliche Erleichterung dar, denn die Wahl des trivialen deterministischen Algorithmus ist sogar fast optimal: Speichere alle Häufigkeiten in einem Array mit  $m$  Zellen. Wenn  $(a_1, \dots, a_m)$  der Häufigkeitsvektor ist, dann genügt ein Speicher von  $\sum_{u=1}^m \log_2(a_u)$  Bits.

Die Berechnung von  $H_1 = \sum_{u \in U} a_u$  ist trivial, die exakte Berechnung von  $H_k$  für  $k \neq 1$  ist aber komplex.

**Satz 4.12** Sei  $\varepsilon < \frac{1}{2}$  beliebig und sei  $A$  ein probabilistischer Algorithmus, der  $H_k$  (für  $k \neq 1$ ) exakt mit Fehler höchstens  $\varepsilon$  im Streaming-Data Modell berechnet. Dann benötigt  $A$  Speicherkomplexität mindestens  $\Omega(m)$ , wenn  $m$  die Anzahl verschiedener Schlüssel ist.

**Beweis:** Wir übernehmen das Argument aus Satz 4.11 und setzen  $m^* = |x| + |y|$ . Diesmal ist zu beachten, dass

$$D_m(x, y) = \begin{cases} 1 & H_0 = m^* \\ 0 & H_0 < m^*, \end{cases}$$

beziehungsweise für  $k > 1$

$$D_m(x, y) = \begin{cases} 1 & H_k = m^* \\ 0 & H_k > m^* \end{cases}$$

und der Rest des Arguments folgt analog.  $\square$

Wir haben also nur dann eine Chance speicher-effizient zu arbeiten, wenn wir approximative Berechnungen zulassen und glücklicherweise gelingt dies, wie wir in den nachfolgenden Abschnitten sehen werden.

---

### Aufgabe 36

Häufig sollen Datenströme auf *Unregelmäßigkeiten* hin überwacht werden. Eine einfaches Kriterium in diesem Kontext ist die Frage, ob sich die relative Häufigkeit eines Elementes in einem Datenstrom stark verändert. Bei der Kommunikationsversion des Problems erhält Alice einen Vektor  $X = (x_1, \dots, x_s)$  und Bob einen Vektor  $Y = (y_1, \dots, y_r)$ , wobei die  $x_i$  und die  $y_i$  aus derselben Schlüsselmenge  $S$  kommen. Als Änderungsgrad  $G$  eines Schlüssels  $a \in S$  definieren wir:

$$G(a) = \begin{cases} 1 & \text{falls } a \text{ weder in } X \text{ noch in } Y \text{ vorkommt,} \\ \infty & \text{falls } a \text{ entweder in } X \text{ oder in } Y \text{ vorkommt,} \\ \max \left\{ \frac{r \cdot |\{i | x_i = a\}|}{s \cdot |\{i | y_i = a\}|}, \frac{s \cdot |\{i | y_i = a\}|}{r \cdot |\{i | x_i = a\}|} \right\} & \text{sonst.} \end{cases}$$

Im *Variationsproblem* für  $\alpha$  soll entschieden werden, ob alle Schlüssel einen Änderungsgrad von höchstens  $\alpha$  besitzen. Wir wollen zeigen, dass das Variationsproblem für jedes feste  $\alpha > 1$  eine Kommunikationskomplexität von  $\Omega(s)$  besitzt.

- Reduziere das Identitätsproblem auf das Variationsproblem.
  - Reduziere das Disjunktheitsproblem auf das Variationsproblem.
  - Warum stellt Teil b) die stärkere Aussage dar?
- 

### 4.2.2 Die Bestimmung von $H_0$

Wenn wir annehmen, dass der Hauptspeicher alle  $n$  tatsächlichen wie auch alle  $m$  möglichen Daten aufnehmen kann, dann erlaubt der Bitvektor-Ansatz<sup>3</sup> eine Lösung in Linearzeit. Wenn die Anzahl  $m$  der möglichen Schlüssel zu groß ist, aber die Anzahl der tatsächlich vorkommenden Schlüssel hinreichend klein ist, dann wird man Hashing durchführen und bei entsprechender Größe der Hash-Tabelle eine Lösung in erwarteter linearer Zeit erhalten.

Im Sekundärspeichermode ist das oberste Gebot eine Minimierung der Anzahl der Zugriffe. Aus der Vorlesung Theoretische Informatik 1 ist bekannt, dass vergleichsorientierte Algorithmen mindestens  $\Omega(n \log_2 n)$  Vergleiche durchführen müssen und deshalb ist Merge-Sort eine gute Wahl.

Im Streaming-Data Modell benötigt nach Satz 4.1 jeder probabilistische Algorithmus, der die Anzahl der verschiedenen Schlüssel exakt bestimmt, mindestens Speicherkomplexität  $\Omega(m)$ .

---

<sup>3</sup>Für jedes  $i$  wird  $\text{BIT}[x_i] = 1$  gesetzt, wobei anfänglich  $\text{BIT} \equiv 0$ .

Wenn wir also vernünftige Speicheranforderungen im Streaming-Data Modell erhalten wollen, dann müssen wir approximative Lösungen erlauben.

Wir benutzen Hashing, da bei  $s$  verschiedenen Schlüsseln höchstens  $s$  verschiedene Hashwerte berechnet werden. Die Anzahl besetzter Hashzellen sollte somit zumindest schwache Rückschlüsse auf die tatsächliche Anzahl verschiedener Schlüssel erlauben.

#### Algorithmus 4.13 Approximieren durch Hashing

- (1) Wähle eine natürliche Zahl  $T > 1$  und eine zufällige Hashfunktion  $h : U \rightarrow \{1, \dots, T\}$ . Setze die boolesche Variable GROSS auf falsch.
- (2) Durchlaufe die Eingabe  $(x_1, \dots, x_n)$ : Wenn  $h(x_i) = T$  für ein  $i$ , dann setze GROSS auf wahr.
- (3) Gib die Antwort GROSS aus.

Beachte, dass wir nur ein Speicherbit benötigen. Wir hoffen, eine Unterscheidung zwischen *höchstens*  $T$  und *mindestens*  $(1 + \varepsilon) \cdot T$  verschiedenen Schlüsseln zu erreichen. Da bei  $d$  verschiedenen Schlüsseln die Hashzelle  $T$  mit Wahrscheinlichkeit  $(1 - \frac{1}{T})^d$  leer bleibt, ergibt sich das folgende Resultat.

**Lemma 4.14** *Wenn eine Eingabefolge  $d \leq T$  verschiedene Schlüssel besitzt, dann wird Algorithmus 4.13 die Antwort GROSS = falsch mit Wahrscheinlichkeit  $(1 - \frac{1}{T})^d \geq (1 - \frac{1}{T})^T$  ausgeben.*

*Gibt es andererseits  $d \geq (1 + \varepsilon) \cdot T$  verschiedene Schlüssel, dann wird Algorithmus 4.13 die Antwort GROSS = falsch mit Wahrscheinlichkeit  $(1 - \frac{1}{T})^d \leq (1 - \frac{1}{T})^{(1+\varepsilon) \cdot T}$  ausgeben.*

Wir haben damit nur eine schwache Trennung zwischen höchstens  $T$  und mindestens  $(1 + \varepsilon) \cdot T$  verschiedenen Schlüsseln erreicht, aber wir können Algorithmus 4.13 *boosten*: Wir arbeiten wieder mit einem Parameter  $T$ , aber diesmal mit  $k$  zufällig gezogenen Hashfunktionen  $h_1, \dots, h_k : U \rightarrow \{1, \dots, T\}$ .

Angenommen, es liegen höchstens  $T$  verschiedene Schlüssel vor. Dann ist die erwartete Häufigkeit der Antwort *falsch* mindestens  $k \cdot (1 - \frac{1}{T})^T$ , während die erwartete Häufigkeit bei mindestens  $(1 + \varepsilon) \cdot T$  verschiedenen Schlüsseln durch  $k \cdot (1 - \frac{1}{T})^{(1+\varepsilon) \cdot T}$  nach oben beschränkt ist. Wir geben deshalb im Boosting-Ansatz die Ausgabe *falsch*, wenn mindestens

$$S = k \cdot \frac{(1 - \frac{1}{T})^{(1+\varepsilon) \cdot T} + (1 - \frac{1}{T})^T}{2}$$

Hashfunktionen die Hashzelle  $T$  *nicht* besetzen; ansonsten wird die Antwort *wahr* gegeben. Beachte, dass

$$\begin{aligned} k \cdot (1 - \frac{1}{T})^{(1+\varepsilon) \cdot T} \cdot (1 + \frac{(1 - 1/T)^{-\varepsilon \cdot T} - 1}{2}) &= k \cdot (1 - \frac{1}{T})^{(1+\varepsilon) \cdot T} \cdot (\frac{1 + (1 - 1/T)^{-\varepsilon \cdot T}}{2}) \\ &= S \\ &= k \cdot (1 - \frac{1}{T})^T \cdot (\frac{1 + (1 - 1/T)^{\varepsilon \cdot T}}{2}) \\ &= k \cdot (1 - \frac{1}{T})^T \cdot (1 - \frac{1 - (1 - 1/T)^{\varepsilon \cdot T}}{2}). \end{aligned}$$

Wir wenden Lemma 1.2 an und erhalten

$$\frac{(1 - 1/T)^{-\varepsilon \cdot T} - 1}{2} = \Omega(\varepsilon) \quad \text{sowie} \quad \frac{1 - (1 - 1/T)^{\varepsilon \cdot T}}{2} = \Omega(\varepsilon).$$

Die Chernoff Ungleichung besagt jetzt, dass die Fehlerwahrscheinlichkeit, bei höchstens  $T$  verschiedenen Schlüsseln oder bei mindestens  $(1 + \varepsilon) \cdot T$  verschiedenen Schlüsseln, negativ exponentiell durch  $e^{-\Omega(\varepsilon^2 \cdot k)}$  beschränkt ist. Also genügen  $O(\frac{1}{\varepsilon^2} \cdot \ln \frac{1}{\delta})$  Hashfunktionen, um eine Fehlerwahrscheinlichkeit von höchstens  $\delta$  zu erreichen.

Wenn wir die Anzahl verschiedener Schlüssel bis auf den Faktor  $(1 + O(\varepsilon))$  approximieren möchten, dann führen wir das obige Verfahren für jedes  $T \in \{(1 + \varepsilon)^r \mid r \leq \log_{1+\varepsilon} n\}$  mit einer jeweils zufälligen Auswahl von  $k = O(\frac{1}{\varepsilon^2} \cdot \ln(\frac{\log_{1+\varepsilon} n}{\delta}))$  Hashfunktionen durch und erreichen weiterhin eine Fehlerwahrscheinlichkeit von höchstens  $\delta$ . Es ist  $\log_{1+\varepsilon} n = \frac{\ln(n)}{\ln(1+\varepsilon)}$  und mit Lemma 1.2 folgt  $\log_{1+\varepsilon} n = O(\frac{\ln(n)}{\varepsilon})$  und wir erhalten:

**Satz 4.15** *Bei einer zufälligen Auswahl von  $k = O(\frac{1}{\varepsilon^2} \cdot \ln(\frac{\log_{1+\varepsilon} n}{\delta}))$  Hashfunktionen für jedes  $T \in \{(1 + \varepsilon)^r \mid r \leq \log_{1+\varepsilon} n\}$  wird die Anzahl verschiedener Schlüssel mit Wahrscheinlichkeit  $1 - \delta$  bis auf den Faktor  $1 + \varepsilon$  richtig bestimmt. Deshalb genügt die Speicherkomplexität*

$$O(k \cdot \log_{1+\varepsilon} n) = O\left(\frac{\ln(n)}{\varepsilon^3} \cdot \ln\left(\frac{\ln(n)}{\varepsilon \cdot \delta}\right)\right).$$

Ein zweiter, besserer Ansatz basiert wieder auf Stichproben. Wir haben allerdings gesehen, dass eine Stichprobe gleichverteilt gezogener Schlüssel die Anzahl verschiedener Schlüssel möglicherweise mit einem zu großen Fehler bestimmt. Wir versuchen deshalb, eine verlässliche *Stichprobe verschiedener Schlüssel* zu berechnen. Dazu weisen wir jedem gesehenen Schlüssel eine zufällig bestimmte Priorität zu und halten alle Schlüssel ab einer Mindestpriorität als Stichprobe fest. Wenn die Stichprobe überläuft, dann ist die Mindestpriorität um 1 hochzusetzen und alle Schlüssel mit zu kleiner Priorität werden entfernt.

#### Algorithmus 4.16 Bestimmung einer Stichprobe verschiedener Schlüssel

- (1)  $m$  sei eine Zweierpotenz und mindestens so groß wie die Anzahl möglicher verschiedener Schlüssel.

Bestimmung der Prioritäten: Wähle Parameter  $A \in \{1, \dots, m-1\}$  und  $B \in \{0, \dots, m-1\}$  zufällig und definiere die Hashfunktion  $h_{A,B}(u) = (A \cdot u + B) \bmod m$ . Schließlich wähle

$$p(u) = \text{die Anzahl der führenden Nullen in der Binärdarstellung von } h_{A,B}(u)$$

als Prioritätszuweisung. (Wir fordern, dass die Binärdarstellung durch führende Nullen auf die exakte Länge  $\log_2 m$  aufgefüllt wird.)

*Kommentar:* Beachte, dass  $\text{prob}[p(u) \geq k] = 2^{-k}$ .

- (2) Setze PRIORITÄT = 0 und STICHPROBE =  $\emptyset$ .  $S$  sei die Maximalgröße einer Stichprobe. (Wähle  $S$  so groß wie möglich. Allerdings ist  $S$  durch den verfügbaren Speicher nach oben beschränkt, denn die Anzahl verschiedener Schlüssel in STICHPROBE muss bestimmt werden.)

Wiederhole für  $i = 1, \dots, n$ :

- (2a) Füge  $x_i$  zur Menge STICHPROBE hinzu, falls  $p(x_i) \geq \text{PRIORITÄT}$ .  
 (2b) Solange STICHPROBE mehr als  $S$  Schlüssel besitzt, entferne alle Schlüssel  $x$  mit  $p(x) = \text{PRIORITÄT}$  und erhöhe PRIORITÄT um 1.

(3) Setze  $p = \text{PRIORITÄT}$  und gib  $2^p \cdot |\text{STICHPROBE}|$  als Schätzung aus.

Wir analysieren Algorithmus 4.16. Es ist  $\text{prob}[p(x) \geq p] = 2^{-p}$  und deshalb folgt für jedes fixierte  $p$

$$\begin{aligned} E[|\{x \mid p(x) \geq p \text{ und es gibt } i \text{ mit } x = x_i\}|] &= \sum_{x \text{ ein Schlüssel}} \text{prob}[p(x) \geq p] \\ &= 2^{-p} \cdot \text{Anzahl verschiedener Schlüssel.} \end{aligned}$$

Algorithmus 4.16 setzt  $p = \text{PRIORITÄT}$  und wir erhalten

$$|\{x \mid p(x) \geq p \text{ und es gibt } i \text{ mit } x = x_i\}| = |\text{STICHPROBE}|.$$

Also folgt

$$E[2^p \cdot |\text{STICHPROBE}|] = \text{Anzahl verschiedener Schlüssel.}$$

Die Schätzung in Schritt (3) ist also gut, wenn große Abweichungen vom Erwartungswert  $E[|\text{STICHPROBE}|]$  nicht wahrscheinlich sind.

---

#### Aufgabe 37

Bestimme die Varianz  $V[|\text{STICHPROBE}|]$  und zeige, dass  $V[|\text{STICHPROBE}|] \leq E[|\text{STICHPROBE}|]$  gilt.

---

Wir setzen  $E = E[|\text{STICHPROBE}|]$  und die Tschebyscheff Ungleichung liefert somit

$$\text{prob}[|E - |\text{STICHPROBE}|| > \varepsilon \cdot E] = O\left(\frac{E}{\varepsilon^2 \cdot E^2}\right) = O\left(\frac{1}{\varepsilon^2 \cdot E}\right).$$

Sei  $V$  die Anzahl der tatsächlich auftretenden verschiedenen Schlüssel. Dann ist  $V = 2^p \cdot E$  und wir erhalten

$$\begin{aligned} \text{prob}[|V - 2^p \cdot |\text{STICHPROBE}|| > \varepsilon \cdot V] &= \text{prob}[2^p \cdot |E - |\text{STICHPROBE}|| > \varepsilon 2^p \cdot E] \\ &= \text{prob}[|E - |\text{STICHPROBE}|| > \varepsilon \cdot E] \\ &= O\left(\frac{1}{\varepsilon^2 \cdot E}\right) \end{aligned}$$

Natürlich müssen wir einen kleinen Fehler erreichen und müssen deshalb  $E \gg \frac{1}{\varepsilon^2}$  fordern. Dies bedingt eine Stichprobengröße  $S$  mit  $S \gg \frac{1}{\varepsilon^2}$ .

---

#### Aufgabe 38

(a) Wiederhole Algorithmus 4.16  $O(\ln(\frac{1}{\delta}))$ -mal mit maximaler Stichprobengröße  $S = O(\frac{1}{\varepsilon^2})$ . Wir verwenden in jedem Versuch zufällige und unabhängig voneinander gewählte Prioritätszuweisungen. Wir bestimmen alle Stichprobengrößen (nach entsprechender Skalierung mit  $2^p$ ) und geben dann den Median aus.

Zeige: Wir erhalten eine bis auf den Faktor  $1 + \varepsilon$  exakte Schätzung mit Wahrscheinlichkeit  $1 - \delta$ . Also genügt die Speicherplatzkomplexität  $O(\frac{1}{\varepsilon^2} \cdot \ln(\frac{1}{\delta}))$ . Damit ist dieses Verfahren wesentlich besser als das Verfahren aus Satz 4.15.

(b) Was kann passieren, wenn wir  $m$  kleiner als die Anzahl der möglichen verschiedenen Schlüssel wählen?

---

### 4.2.3 Die Bestimmung von $H_2$

Wir nehmen wieder das Streaming-Data Modell an, wobei die Daten  $x_i$  zum Universum  $U = \{1, \dots, m\}$  gehören mögen. Unser Ziel ist die approximative Bestimmung von

$$H_2 = \sum_{u \in U} a_u^2,$$

wobei  $a_u = |\{i \mid x_i = u\}|$  die Häufigkeit des Schlüssels  $u$  ist.

#### Algorithmus 4.17 Sketching $H_2$

- (1) Bestimme eine zufällige Hashfunktion

$$h : U \rightarrow \{-1, 1\}$$

und setze SUMME = 0.

- (2) Wiederhole für alle Daten  $x_i$

Setze SUMME = SUMME +  $h(x_i)$ .

Wir bestimmen zuerst Erwartungswert und Varianz von SUMME<sup>2</sup>.

**Lemma 4.18** Die Hashfunktion  $h$  möge aus einer vierfach unabhängigen Menge von Hashfunktionen gewählt werden. Dann ist

$$E[ \text{SUMME}^2 ] = H_2 \quad \text{und} \quad V[ \text{SUMME}^2 ] \leq 2 \cdot H_2^2.$$

**Beweis:** Es ist

$$\begin{aligned} E[ \text{SUMME}^2 ] &= E[ (\sum_{u=1}^m a_u \cdot h(u))^2 ] \\ &= E[ \sum_{u=1}^m a_u^2 \cdot h(u)^2 ] + E[ \sum_{u \neq v} a_u \cdot a_v \cdot h(u) \cdot h(v) ] \\ &= E[ \sum_{u=1}^m a_u^2 ] = \sum_{u=1}^m a_u^2 = H_2, \end{aligned}$$

denn offensichtlich ist  $h(u)^2 = 1$  für jedes  $u$  und  $E[ h(u) \cdot h(v) ] = 0$  für  $u \neq v$ . Also stimmt der Erwartungswert von SUMME<sup>2</sup> mit  $H_2$  überein. Wir berechnen als nächstes die Varianz  $V[ \text{SUMME}^2 ] = E[ \text{SUMME}^4 ] - E[ \text{SUMME}^2 ]^2$ . Zuerst beachte

$$E[ \text{SUMME}^4 ] = E[ (\sum_{u=1}^m a_u \cdot h(u))^4 ].$$

Beim Ausmultiplizieren treten Terme der Form  $E[ h(u_1) \cdot h(u_2) \cdot h(u_3) \cdot h(u_4) ]$  auf. Da wir vierfache Unabhängigkeit annehmen, verschwindet ein solcher Term genau dann *nicht*, wenn entweder  $u_1 = u_2 = u_3 = u_4$  oder  $u_1 = u_2 \neq u_3 = u_4$  oder  $u_1 = u_3 \neq u_2 = u_4$  oder  $u_1 = u_4 \neq u_2 = u_3$ . Also folgt

$$\begin{aligned} E[ \text{SUMME}^4 ] &= E[ \sum_{u=1}^m a_u^4 \cdot h(u)^4 ] + 2 \cdot 3 \cdot E[ \sum_{u \neq v} a_u^2 \cdot a_v^2 \cdot h(u)^2 \cdot h(v)^2 ] \\ &= \sum_{u=1}^m a_u^4 + 6 \cdot \sum_{u \neq v} a_u^2 \cdot a_v^2 \end{aligned}$$

und wir erhalten die Varianz

$$\begin{aligned} V[\text{SUMME}^2] &= \sum_{u=1}^m a_u^4 + 6 \cdot \sum_{u \neq v} a_u^2 \cdot a_v^2 - \left( \sum_{u=1}^m a_u^2 \right)^2 \\ &= 4 \cdot \sum_{u \neq v} a_u^2 \cdot a_v^2 \leq 2 \cdot H_2^2 \end{aligned}$$

und das war zu zeigen.  $\square$

Wir wenden die Tschebyscheff Ungleichung  $\text{prob}[|X - E[X]| > t] \leq \frac{V[X]}{t^2}$  an und erhalten  $\text{prob}[|\text{SUMME}^2 - E[\text{SUMME}^2]| > t] \leq \frac{V[\text{SUMME}^2]}{t^2}$ . Also ist

$$\text{prob}[|\text{SUMME}^2 - H_2| > \lambda \cdot H_2] \leq \frac{2 \cdot H_2^2}{\lambda^2 \cdot H_2^2} = \frac{2}{\lambda^2}$$

für jedes  $\lambda$  und selbst relativ kleine Abweichungen sind somit leider nicht als unwahrscheinlich ausgeschlossen. Deshalb boosten wir. Ein erster Ansatz wählt  $k$  Hashfunktionen  $h_1, \dots, h_k$  und gibt das Ergebnis

$$\text{SCHÄTZUNG}_k = \frac{\sum_{i=1}^k \text{SUMME}_i^2}{k}$$

aus. Wir gehen wie in Beispiel 1.4 vor und beobachten, dass der Erwartungswert stabil bleibt, aber dass die Varianz um den Faktor  $k$  fällt. Also erhalten wir nach  $k$ -maliger Wiederholung und Durchschnittsbildung

$$\text{prob}[|\text{SCHÄTZUNG}_k - H_2| > \lambda \cdot H_2] \leq \frac{2}{k \cdot \lambda^2}$$

Beachte, dass wir für  $k = \frac{8}{\lambda^2}$  eine durch  $\frac{1}{4}$  nach oben beschränkte Fehlerwahrscheinlichkeit erhalten. Ein zweiter Boosting-Ansatz fordert  $k^* = k \cdot s$  Wiederholungen von Algorithmus 4.17. Die jeweiligen Summenergebnisse werden in  $s$  Klassen von jeweils  $k$  Wiederholungen eingeteilt und für jede Klasse wird eine Schätzung ermittelt. Schließlich bestimmt man den Median  $M_{k^*}$  der  $s$  Schätzungen.

**Satz 4.19** *Es sei  $k = \frac{8}{\lambda^2}$  und  $k^* = k \cdot s$ .*

- (a) *Wenn der arithmetische Durchschnitt  $\text{SCHÄTZUNG}_k$  der Summenwerte berechnet wird, dann ist  $|\text{SCHÄTZUNG}_k - H_2| > \lambda \cdot H_2$  mit Wahrscheinlichkeit höchstens  $\frac{2}{k \cdot \lambda^2}$ .*
- (b) *Wenn der Median  $M_{k^*}$  der Summenwerte ausgegeben wird, dann ist  $|M_{k^*} - H_2| > \lambda \cdot H_2$  mit Wahrscheinlichkeit höchstens  $2^{-\Omega(s)}$ .*
- (c) *Eine bis auf den Faktor  $1 + \varepsilon$  exakte Schätzung von  $H_2$  wird mit Wahrscheinlichkeit mindestens  $1 - \delta$  und Speicherplatzkomplexität  $O(\frac{1}{\varepsilon^2} \cdot \log_2(\frac{1}{\delta}))$  erreicht.*

**Beweis:** Wir haben (a) bereits gezeigt. Für (b) beachten wir, dass eine Schätzung nur mit Wahrscheinlichkeit höchstens  $\frac{1}{4}$  außerhalb des Toleranzintervalls  $|M_{k^*} - H_2| \leq \lambda \cdot H_2$  liegt. Wenn also der Median außerhalb des Toleranzintervalls liegt, dann liegt mindestens die Hälfte aller Einzelschätzungen außerhalb und damit mindestens doppelt so viele Einzelschätzungen wie erwartet. Das Ergebnis folgt jetzt aus der Chernoff-Schranke.  $\square$

### Aufgabe 39

In Anlehnung an das Schema der  $H_2$ -Schätzung beschreiben wir den folgenden Algorithmus für die Bestimmung von  $H_3$ :

- (1) Bestimme eine zufällige Hashfunktion

$$h : U \rightarrow \{1, e^{\frac{2}{3}i\pi}, e^{\frac{4}{3}i\pi}\}$$

/\* Es wird also auf eine der komplexen dritten Einheitswurzeln gehasht. \*/

- (2) SUMME := 0;

- (3) Wiederhole für alle Daten
- $x_i$

$$\text{SUMME} := \text{SUMME} + h(x_i)$$

- a) Es sei  $Y = \text{SUMME}^3$ . Zeige  $E[Y] = H_3$ .
- b) Warum kann dieses Ergebnis nicht benutzt werden, um eine gute Schätzung von  $H_3$  mit geringem Speicheraufwand zu erhalten?

### 4.3 Häufigkeitsanfragen

Verkehrsmessungen im Internet, also Häufigkeitsmessungen für Flüsse<sup>4</sup>, sind besonders datenintensiv: Bereits heute können DRAMs nicht mit der Geschwindigkeit der Links mithalten und diese Lücke wird eher anwachsen, da die Geschwindigkeit der DRAMs jährlich um höchstens 10%, die Linkgeschwindigkeit sich aber momentan jährlich sogar verdoppelt. DRAM-Zähler können somit Häufigkeitsmessungen nicht exakt durchführen, vielmehr müssen Messungen stichprobenartig, wie etwa in CISCO's NetFlow Algorithmus, durchgeführt werden.

Der NetFlow Algorithmus ist aber aus zwei Gründen nicht überzeugend. Zuerst sind die Schätzungen nicht präzise und insbesondere werden keine fehlerfreien unteren Schranken bestimmt: Sicherlich wird man nicht willens sein, einen überhöht angesetzten Paketverkehr zu bezahlen. Zuletzt ist die Speicherplatzkomplexität immens und es ist nicht ungewöhnlich, dass nur 10% der erfassten Daten ausgewertet werden.

Aus diesen Gründen konzentriert man sich mehr und mehr auf die "heavy hitter", also Flüsse mit großer Häufigkeit. Diese Maßnahme wird zum Beispiel auch durch die Dominanz großer Flüsse unterstützt, da ein kleiner Prozentsatz (ungefähr 10%) aller Flüsse einen Großteil des Paketverkehrs (ungefähr 90%) ausmacht [FP].

Wir beschreiben Algorithmen für die Bestimmung der heavy hitter, also für die Bestimmung häufiger Schlüssel, in den beiden nächsten Abschnitten. In Abschnitt 4.3.3 zeigen wir unter Anderem wie man die Anzahl seltener Flüsse abschätzt, solange seltene Flüsse für einen signifikanten Anteil des Verkehrsaufkommens verantwortlich sind.

#### 4.3.1 Heavy Hitters

Wir haben bereits in Satz 4.11 gesehen, dass die Bestimmung des häufigsten Schlüssels, bzw. der Häufigkeit eines häufigsten Schlüssels, die Speicherkomplexität  $\Omega(m)$  verlangt, selbst wenn wir mit randomisierten Algorithmen arbeiten und nur eine approximative Bestimmung anstreben. Wie sieht es aus, wenn wir die Problemstellung verändern und zu einem Schwellenwert  $\theta$  nur alle Schlüssel mit Häufigkeit größer als  $\theta \cdot n$  bestimmen wollen?

##### Aufgabe 40

$A$  sei ein deterministischer Algorithmus. Wenn  $m$  die Anzahl verschiedener Schlüssel unter den ersten  $n$  Schlüsseln ist, dann wird  $A$  für die Bestimmung aller Schlüssel mit Häufigkeit größer als  $\frac{n}{2}$  mindestens die Speicherkomplexität  $\Omega(m \cdot \log_2(\frac{n}{m}))$  benötigen.

<sup>4</sup>Ein Fluß ist eine Paketmenge mit fixiertem Sender und Empfänger. Zusätzlich muss der (optionale) "Flußname" identisch sein.

Hinweis: Wende das Kommunikationsmodell an. Benutze das Fooling Argument, d.h. zeige, dass Alice für bestimmte verschiedene Eingaben keine identischen Nachrichten verschicken darf. Um den entsprechenden Nachweis zu führen, konstruiere „passende“ Eingaben für Bob. Die Ungleichung  $\binom{a}{b} \geq (\frac{a}{b})^b$  kann benutzt werden.

Also ist auch hier keine substantielle Speicherplatz-Ersparnis im Vergleich zu  $\Omega(m)$  möglich. Glücklicherweise wird das Häufigkeitsproblem plötzlich machbar, wenn wir nur verlangen, dass eine Menge  $K$  von höchstens  $\frac{1}{\theta}$  Schlüsseln<sup>5</sup> berechnet wird, unter denen sich alle Schlüssel mit Häufigkeit größer als  $\theta \cdot n$  befinden müssen.

Die Grundidee steckt schon in der seit Langem bekannten Lösung des Mehrheitsproblems, nämlich der Bestimmung eines Schlüssels mit Häufigkeit größer als  $\frac{n}{2}$ . Eine Lösung entfernt Paare *verschiedener* Schlüssel. In diesem Prozess verliert der Mehrheitschlüssel nie seine Mehrheitseigenschaft und wird bis zuletzt überleben. Ein zweiter Datendurchlauf kann dann die Mehrheitseigenschaft überprüfen.

#### Algorithmus 4.20 Bestimmung häufiger Schlüssel

- (1) Der Datenstrom bestehe aus dem Vektor  $x = (x_1, x_2, \dots, x_n)$  mit  $x_i \in \{1, \dots, m\}$ .
- (2) Die Menge  $K$  ist anfänglich leer und das Array “Zähler” ist auf Null initialisiert.
- (3) For  $i = 1$  to  $n$  do
  - (3a) Wenn  $x_i \in K$ , dann erhöhe den Zähler von  $x_i$  um Eins.
  - (3b) Wenn  $x_i \notin K$ , dann füge  $x_i$  zu  $K$  hinzu und setze den Zähler von  $x_i$  auf Eins. Wenn  $K$  jetzt genau  $\frac{1}{\theta}$  Elemente besitzt, dann erniedrige alle Zähler um Eins und entferne alle Schlüssel mit Zählerstand Null.

**Satz 4.21** *Algorithmus 4.20 bestimmt eine Menge  $K$  von höchstens  $\frac{1}{\theta}$  Schlüsseln, unter denen sich alle Schlüssel mit Häufigkeit größer als  $\theta \cdot n$  befinden. Der Algorithmus rechnet in Linearzeit und die Menge  $K$  besteht aus höchstens  $\theta$  Schlüsseln.*

**Beweis:** Übungsaufgabe. □

Die Speicherplatzkomplexität von Algorithmus 4.20 kann selbst von randomisierten Algorithmen nicht wesentlich unterschritten werden, da Speicherplatz  $\Omega(\frac{1}{\theta} \cdot \log_2(\theta \cdot n))$  schon benötigt wird, um die  $\binom{n}{1/\theta}$  möglichen Mengen der heavy hitter darzustellen. Allerdings werden keine Schätzungen der individuellen Häufigkeiten berechnet und zusätzlich werden möglicherweise Schlüssel mit sehr geringer Häufigkeit ausgegeben. Die letzte Schwäche können wir mit dem randomisierten *Sample-and-Count* Algorithmus [EV] zumindest teilweise beseitigen: Der Algorithmus nimmt einen Schlüssel nur mit sehr kleiner Wahrscheinlichkeit in seine Stichprobe auf, ein heavy hitter erhält aber entsprechend viele Versuche und wird deshalb hochwahrscheinlich erfasst.

#### Aufgabe 41

Wir beschreiben Algorithmus Sample-and-Count, der mit einer Stichprobe  $S$  arbeitet. Wenn  $x_n$  ein Schlüssel des Datenstroms ist, dann führe die folgende Fallunterscheidung durch:

- (1) Wenn  $x_n \notin S$ , dann füge  $x_n$  zu  $S$  mit Wahrscheinlichkeit  $\frac{r}{n}$  hinzu. Initialisiere einen Zähler für  $x_n$  mit dem Wert 1.  
/\*  $r$  wird später bestimmt. \*/

<sup>5</sup>Offensichtlich gibt es höchstens  $\frac{1}{\theta}$  Schlüssel mit Häufigkeit mindestens  $\theta \cdot n$ .

- (2) Wenn  $x_n \in S$ , dann erhöhe den Zähler von  $x_n$  um Eins.  
 (3) Gib alle Schlüssel in  $S$  aus, deren Zählerwert mindestens  $(\theta - \varepsilon) \cdot n$  beträgt.  
 /\* Es muss  $\varepsilon \leq \theta$  gelten. \*/

- (a) Angenommen der Schlüssel  $x$  ist ein heavy hitter, d.h.  $x$  tritt mit Häufigkeit mindestens  $\theta \cdot n$  auf. Zeige, dass  $x$  mit Wahrscheinlichkeit höchstens  $(1 - \frac{\varepsilon}{n})^{\varepsilon \cdot n} \leq e^{-r \cdot \varepsilon}$  nicht in Schritt (3) ausgegeben wird. Die Wahrscheinlichkeit, dass irgendein heavy hitter ausgelassen wird, ist damit höchstens  $\frac{1}{\theta} \cdot e^{-r \cdot \varepsilon}$ .  
 (b) Nur Schlüssel mit Häufigkeit mindestens  $(\theta - \varepsilon) \cdot n$  werden ausgegeben. Bestimme  $r$ , so dass jeder heavy hitter mit Wahrscheinlichkeit mindestens  $1 - \delta$  ausgegeben wird. Zeige, dass die Reaktionszeit pro Schlüssel konstant ist und dass die erwartete Größe der Stichprobe  $S$  durch  $O(\frac{1}{\varepsilon} \cdot \log_2(\frac{1}{\delta \cdot \theta}))$  beschränkt ist.

Beachte, dass alle Pakete, die zu einem in der Stichprobe befindlichen Fluss gehören, bearbeitet werden müssen, um die fehlerfreie untere Schranke  $(\theta - \varepsilon) \cdot n$  zu erhalten. Deshalb muss mit dem schnellen SRAM-Speicher gearbeitet werden. Da wir uns aber auf heavy hitters beschränken, ist ein relativ kleiner SRAM-Speicher ausreichend.

Im nächsten Abschnitt wenden wir Bloom-Filter auf die Bestimmung der heavy hitter an. Während der Zeitaufwand für die Verarbeitung eines Pakets im Vergleich mit sample-and-count leicht ansteigt, wird jeder heavy hitter mit Wahrscheinlichkeit 1 erfasst und wir erhalten vergleichbare Schätzungen des tatsächlichen Paketaufkommens, falls wenige Flüsse stark dominieren.

#### Aufgabe 42

Wir stellen ein weiteres Schätzverfahren vor. Wir arbeiten wieder mit einer anfänglich leeren Stichprobe  $S$  und benutzen diesmal Zeitintervalle der Länge  $\frac{1}{\varepsilon}$ .

- (1) Alle während eines Zeitfensters  $Z$  auftretenden Schlüssel werden, wenn nicht schon zu  $S$  gehörig, in die Stichprobe eingefügt und ihre Zähler werden aktualisiert.  
 (2) Nach Ablauf von  $Z$  werden alle Zählerstände um Eins verringert und alle Schlüssel mit Häufigkeit 0 werden aus  $S$  entfernt.  
 (3) Alle Schlüssel mit Häufigkeit mindestens  $(\theta - \varepsilon) \cdot n$  werden ausgegeben.  
 (a) Jeder Schlüssel mit Häufigkeit mindestens  $\theta \cdot n$  wird ausgegeben und alle ausgegebenen Schlüssel haben die Häufigkeit mindestens  $(\theta - \varepsilon) \cdot n$ .  
 (b) Die Reaktionszeit pro Schlüssel ist konstant und die Größe der Stichprobe ist durch  $O(\frac{1}{\varepsilon} \cdot \log_2(\varepsilon \cdot n))$  beschränkt.

### 4.3.2 Der Bloom-Filter Sketch

Wie verallgemeinern unser Häufigkeitsmodell und nehmen an, dass der Datenstrom die Form  $((x_i, c_i) \mid i)$  hat. Unser Ziel ist die Beantwortung der verallgemeinerten Häufigkeitsanfrage für Schlüssel  $x$ , die zum Zeitpunkt  $i$  die Antwort

$$H(x) = \sum_{k \leq i, x_k = x} c_k$$

besitzt: Wir summieren also die Inkremente  $c_k$  für die Zeitpunkte  $k \leq i$ , für die der Schlüssel  $x$  im Datenstrom erschienen ist. Wenn stets  $c_i = 1$ , dann erhalten wir das konventionelle Häufigkeitsmodell und die konventionellen Häufigkeitsanfragen, wir werden aber im Folgenden nur  $c_i \geq 0$  annehmen.

Wir möchten eine „Ungenauigkeit“ von höchstens  $\varepsilon$  mit Wahrscheinlichkeit mindestens  $1 - \delta$  erreichen und arbeiten mit  $k = \lceil \ln(\frac{1}{\delta}) \rceil$  Hashfunktionen  $h_1, \dots, h_k : U \rightarrow \{1, \dots, w\}$  für  $w = \frac{\varepsilon}{\delta}$  Zellen. Wir verwenden zählende Bloom-Filter, die den einzelnen Hashfunktionen disjunkte Teilarrays  $Z_1, \dots, Z_k$  des Arrays  $Z = (Z_1, \dots, Z_k)$  zuweisen; jedes Teilarray besteht aus  $w$  Zellen.

**Algorithmus 4.22 Der Bloom-Filter Sketch.**

- (1) Die Arrays  $Z_1, \dots, Z_k$  sind alle zu Null initialisiert.  
 (2) Wiederhole: Wenn das Paar  $(x_i, c_i)$  empfangen wird, dann setze für jedes  $j \leq k$

$$Z_j[h_j(x_i)] = Z_j[h_j(x_i)] + c_i.$$

- (3) Eine verallgemeinerte Häufigkeitsfrage für Schlüssel  $x$  wird mit

$$H'(x) = \min_{1 \leq j \leq k} Z_j[h_j(x)]$$

beantwortet.

**Satz 4.23** *Es gelte  $c_i \geq 0$  für alle  $i$ . Dann ist stets  $H(x) \leq H'(x)$ . Für  $k = \lceil \ln(\frac{1}{\delta}) \rceil$  und  $w = \frac{\varepsilon}{\delta}$  gilt*

$$H'(x) \leq H(x) + \varepsilon \cdot \sum_{y \in U} H(y)$$

*mit Wahrscheinlichkeit mindestens  $1 - \delta$ . Insgesamt werden höchstens  $O(\frac{1}{\varepsilon} \cdot \ln(\frac{1}{\delta}))$  Zellen benutzt und die Laufzeit pro Anfrage ist höchstens  $O(\ln(\frac{1}{\delta}))$ .*

**Beweis:** Es ist stets  $H(x) \leq Z_j[h_j(x)]$  und  $H(x) \leq H'(x) = \min_{j \leq k} Z_j[h_j(x)]$  folgt. Also bleibt nur die Frage, mit welcher Wahrscheinlichkeit  $H'(x)$  zu groß wird. Wir definieren die Zufallsvariable

$$K_j^x = \sum_{y \in U \text{ mit } x \neq y \text{ und } h_j(x) = h_j(y)} H(y)$$

und erhalten

$$\begin{aligned} \text{prob}[H'(x) > H(x) + \varepsilon \cdot \sum_{y \in U} H(y)] &= \text{prob}[\forall j \leq k : Z_j[h_j(x)] > H(x) + \varepsilon \cdot \sum_{y \in U} H(y)] \\ &= \text{prob}[\forall j \leq k : H(x) + K_j^x > H(x) + \varepsilon \cdot \sum_{y \in U} H(y)] \\ &= \text{prob}[\forall j \leq k : K_j^x > \varepsilon \cdot \sum_{y \in U} H(y)] \\ &= (\text{prob}[K_j^x > \varepsilon \cdot \sum_{y \in U} H(y)])^k. \end{aligned} \quad (4.1)$$

Im letzten Schritt haben wir ausgenutzt, dass wir die Hashfunktionen zufällig voneinander ausgewählt haben: An dieser Stelle zahlt sich der Bloom-Filter Ansatz aus.

Wie groß kann  $K_j^x$  werden? Wir definieren die Null-Eins Zufallsvariable  $K_j^x(y)$ , wobei  $K_j^x(y)$  genau dann den Wert Eins annimmt, wenn  $h_j(x) = h_j(y)$ . Wir beachten, dass  $E[K_j^x(y)] \leq 1/w$  für  $x \neq y$  gilt und können jetzt den Erwartungswert von  $K_j^x$  abschätzen:

$$\begin{aligned} E[K_j^x] &= E\left[\sum_{y \in U \text{ mit } x \neq y \text{ und } h_j(x) = h_j(y)} H(y)\right] = \sum_{y \in U \text{ mit } x \neq y} H(y) \cdot E[K_j^x(y)] \\ &\leq \sum_{y \in U \text{ mit } x \neq y} \frac{H(y)}{w} \leq \frac{\varepsilon}{e} \cdot \sum_{y \in U} H(y). \end{aligned}$$

Wir können jetzt die Markoff Ungleichung<sup>6</sup> auf (4.1) anwenden

$$\begin{aligned} \text{prob}[H'(x) > H(x) + \varepsilon \cdot \sum_{y \in U} H(y)] &= (\text{prob}[K_j^x > \varepsilon \cdot \sum_{y \in U} H(y)])^k \\ &\leq \left(\frac{\varepsilon}{e} \cdot \sum_{y \in U} H(y) / \varepsilon \cdot \sum_{y \in U} H(y)\right)^k = e^{-k} \leq \delta \end{aligned}$$

und das war zu zeigen.  $\square$

---

#### Aufgabe 43

Wir erlauben jetzt auch negative "Inkrement"  $c_i$ . Modifiziere Schritt (3) des Bloom-Filter Ansatzes, so dass für die neue Schätzung  $H''(x)$

$$\text{prob}[H(x) - s \cdot \varepsilon \cdot \sum_{y \in U} H(y) \leq H''(x) \leq H(x) + s \cdot \varepsilon \cdot \sum_{y \in U} H(y)] \geq 1 - \delta^{1/s}$$

für konstantes  $s$  gilt.

---

**Bemerkung 4.2** In [EV] werden Bloom-Filter mit sample-and-count und Cisco's NetFlow verglichen. Bloom Filter werden allerdings leicht modifiziert. Zum Einen wird eine *konservative* Aktualisierung durchgeführt: Wenn ein Paket des Flusses  $F$  zu verarbeiten ist, dann wird zuerst der bisherige minimale Wert  $M_F$  von  $F$  festgestellt. Sodann wird der Zählerstand in einer jeden Position  $h_i(x)$  nur dann inkrementiert, wenn der bisherige Wert für  $h_i(x)$  höchstens  $F_M$  beträgt: Der Fluss kann bei einem Wert größer als  $M$  nicht allein für den Zählerwert verantwortlich gewesen sein!

Zum Anderen wird die Methode des *Shielding* angewandt: Es werden Meßintervalle eingeführt und die Pakete von Flüssen, die für einen längeren Zeitraum in jedem Meßintervall häufig vertreten sind, werden direkt gezählt ohne durch den Bloom-Filter laufen zu müssen.

Nach diesen Maßnahmen stellen sich Bloom-Filter und sample-and-count als wesentlich exakter in der Schätzung der heavy hitter heraus, während natürlich NetFlow exaktere Schätzungen der leichteren Flüsse liefert, aber dabei enorme Datenmengen produziert. Der Vergleich zwischen Bloom-Filtern und sample-and-hold wird mit leichten Vorteilen vom Bloom-Filter Sketch gewonnen.

Wir starten eine weitere Schätzung mit den  $-1/1$ -wertigen Hashfunktionen aus Abschnitt 4.2.3.

---

#### Aufgabe 44

Der Datenstrom  $(x_i \mid i)$  sei gegeben. Wir wählen zufällig  $k$  Hashfunktionen  $h_1, \dots, h_k : U \rightarrow \{-1, +1\}$  aus und richten  $k$  anfänglich auf Null gesetzte Zähler  $Z_1, \dots, Z_k$  ein.

Für jeden Schlüssel  $x_i$  des Datenstroms und für jedes  $j \leq k$  setze  $Z_j := Z_j + h_j(x_i)$ . Anschließend geben wir für jeden Schlüssel  $x \in U$  den Wert  $H'(x) = \frac{1}{k} \cdot \sum_{i=1}^k Z_i \cdot h_i(x)$  als Schätzung der Häufigkeit aus.

Sei  $H(x)$  die tatsächliche Häufigkeit des Schlüssels  $x$ .

(a) Zeige, dass  $E\left(\frac{1}{k} \cdot \sum_{i=1}^k z_i \cdot h_i(x)\right) = H(x)$  gilt.

(b) Zeige, dass  $\text{Varianz}\left(\frac{1}{k} \cdot \sum_{i=1}^k z_i \cdot h_i(x)\right) = \frac{1}{k} \cdot \sum_{y \in U, x \neq y} H(y)^2$  gilt.

(c) Wir setzen  $\|H\| = \sqrt{\sum_{y \in U} H(y)^2}$ . Zeige, dass eine Schätzung  $H''(x)$  mit Speicherplatzkomplexität  $O\left(\frac{1}{\varepsilon^2} \cdot \ln\left(\frac{1}{\delta}\right)\right)$  berechnet werden kann, so dass

$$\text{prob}[H(x) - \varepsilon \cdot \|H\| \leq H''(x) \leq H(x) + \varepsilon \cdot \|H\|] \geq 1 - \delta$$

gilt. Hinweis: Benutze das Verfahren aus Satz 4.19.

---

<sup>6</sup> $\text{prob}[X > a] \leq \frac{E[X]}{a}$  für  $X \geq 0$

**Bemerkung 4.3** Die Speicherplatzkomplexität für das Sketching mit  $-1/1$  Hashfunktionen ist also höher als für das Sketching mit Bloom-Filtern. Andererseits wird eine schärfere Approximation erreicht, wenn  $\|H\| \ll \sum_{y \in U} H(y)$ . Die beiden Ansätze sind also nicht direkt vergleichbar.

Allerdings treiben heavy hitters die  $L_2$ -Norm  $\|H\|$  hoch und Bloom-Filter werden in Routing-Anwendungen die Nase vorn haben.

### 4.3.3 Häufigkeitseigenschaften

Wir betrachten Schlüssel-Eigenschaften, die nur von der Häufigkeit abhängen mit der der Schlüssel in einem Datenstrom  $(x_i | i)$  auftritt. “Schlüssel mit Häufigkeit höchstens  $K$ ” oder “Schlüssel mit Häufigkeit mindestens  $\theta \cdot n$ ” sind Beispiele solcher Eigenschaften.

Um die Klasse dieser Eigenschaften zu formalisieren, betrachten wir eine Teilmenge  $E_n$  von  $\{1, \dots, n\}$ . Wir sagen, dass ein Schlüssel  $x$  die Eigenschaft  $E_n$  hat, falls  $|\{j \leq n \mid x_j = x\}| \in E_n$  gilt. Die Menge  $E_n = \{1, \dots, K\}$  formalisiert also die Eigenschaft “höchstens  $K$ -mal aufzutreten”.

**Definition 4.24** Sei  $(x_i | i)$  ein Datenstrom,  $V_n = \{x_j \mid j \leq n\}$  die Menge der verschiedenen Schlüssel  $x_j$  für  $j \leq n$  und sei

$$E_n^* = \{x_j \mid j \leq n \text{ und } x_j \text{ hat Eigenschaft } E_n\}$$

die Menge der Schlüssel mit Eigenschaft  $E_n$ . Dann hat Eigenschaft  $E_n$  die Wahrscheinlichkeit

$$p[E_n] = \frac{|E_n^*|}{|V_n|}.$$

$p[E_n]$  misst also wieviele der verschiedenen Schlüssel im Präfix  $(x_j \mid j \leq n)$  die Eigenschaft  $E_n$  besitzen.

Für den Jaccard-Koeffizienten  $J(E_n^*, V_n) = \frac{|E_n^* \cap V_n|}{|E_n^* \cup V_n|}$  gilt  $J(E_n^*, V_n) = \frac{|E_n^*|}{|V_n|} = p[E_n^*]$ , da  $E_n^*$  eine Teilmenge von  $V_n$  ist. Wir haben Min-Hashing benutzt, um den Jaccard-Koeffizienten  $J(E_n^*, V_n)$  zu berechnen, und dazu insbesondere die Darstellung

$$J(E_n^*, V_n) = \frac{|\{\pi \mid \min_{\pi}(E_n^*) = \min_{\pi}(V_n)\}|}{|\{\pi \mid \pi \text{ ist eine Permutation von } \{1, \dots, n\}\}|}$$

hergeleitet. Also folgt:

$$\pi^{-1}(1) \in E_n^* \text{ gilt mit Wahrscheinlichkeit } J(E_n^*, V_n) = p[E_n^*].$$

### Algorithmus 4.25 Häufigkeitsmessung für Eigenschaften

- (1) Wähle  $k$  zufällige Permutationen  $\pi_1, \dots, \pi_k$  und setze  $\min_j(0) = \infty$  sowie  $\text{Zähler}_j(0) = 0$  für alle  $j \leq k$ .

/\*  $k$  wird später bestimmt. \*/

- (2) Wenn  $x_{i+1}$  zu verarbeiten ist, dann wiederhole für jedes  $j \leq k$

- (a) Berechne  $\pi_j(x_{i+1})$ .

- (b) Wenn  $\pi_j(x_{i+1}) < \min_j(i)$ , dann setze  $\min_j(i+1) = \pi_j(x_{i+1})$  und Zähler $_j(i+1) = 1$ .  
/\* Der Schlüssel  $x_{i+1}$  ist zum ersten Mal aufgetreten und sein Zähler wird deshalb auf Eins gesetzt. \*/
- (c) Wenn  $\pi_j(x_{i+1}) = \min_j(i)$ , dann setze Zähler $_j(i+1) = \text{Zähler}_j(i) + 1$ .  
/\* Ein weiteres Vorkommen des minimalen Schlüssels  $x_{i+1}$  für  $\pi_j$  wurde beobachtet und dementsprechend wird sein Zähler inkrementiert. \*/
- (3) Bestimme die Anzahl  $k^*$  der Schlüssel  $\min_j(n)$  für die Eigenschaft  $E_n$  gilt und gib  $k^*/k$  als Schätzung für die Wahrscheinlichkeit von  $E_n$  aus.

**Aufgabe 45**

Warum können wir weder Reservoir Sampling (Algorithmus 4.1) noch Stichproben verschiedener Elemente (Algorithmus 4.16) anwenden?

Für die Analyse von Algorithmus 4.25 definieren wir die Zufallsvariablen  $X_j$  durch

$$X_j = \begin{cases} 1 & \pi_j^{-1}(1) \in E_n^* \\ 0 & \text{sonst} \end{cases}$$

und berechnen den Erwartungswert  $E[\sum_{j=1}^k X_j] = \sum_{j=1}^k E[X_j] = k \cdot p[E_n]$ . Da die Zufallsvariablen unabhängig voneinander sind, können wir die Chernoff-Schranke anwenden und erhalten

$$\text{prob}\left[ \left| \sum_{j=1}^k X_j - k \cdot p[E_n] \right| \geq \varepsilon \cdot k \cdot p[E_n] \right] \leq 2 \cdot e^{-k \cdot p[E_n] \cdot \varepsilon^2/3}.$$

Also folgt

$$\begin{aligned} \text{prob}\left[ |k^*/k - p[E_n]| \geq \varepsilon \cdot p[E_n] \right] &= \text{prob}\left[ |k^* - k \cdot p[E_n]| \geq \varepsilon \cdot k \cdot p[E_n] \right] \\ &\leq 2 \cdot e^{-k \cdot p[E_n] \cdot \varepsilon^2/3}. \end{aligned}$$

Wenn wir also eine  $\varepsilon$ -Approximation mit Fehlerwahrscheinlichkeit höchstens  $\delta$  fordern, werden wir auf die Bedingung

$$2 \cdot e^{-k \cdot p[E_n] \cdot \varepsilon^2/3} \leq \delta$$

geführt und deshalb auf die Bedingung  $-k \cdot p[E_n] \cdot \varepsilon^2/3 \leq \ln(\frac{\delta}{2})$ , beziehungsweise auf  $k \geq \frac{3}{\varepsilon^2 \cdot p[E_n]} \cdot \ln(\frac{2}{\delta})$ .

**Satz 4.26** Für  $k = O(\frac{1}{\varepsilon^2 \cdot p[E_n]} \cdot \ln(\frac{1}{\delta}))$  erreicht Algorithmus 4.25 eine  $\varepsilon$ -Approximation von  $p[E_n]$  mit Wahrscheinlichkeit mindestens  $1 - \delta$ . Die Laufzeit pro Schlüssel ist durch  $O(k)$  und die Speicherplatzkomplexität durch  $O(k \cdot \log_2 n)$  beschränkt, wenn der für die Permutationen benötigte Speicherplatz nicht gezählt wird.

Wir können also die Anzahl seltener oder häufiger Schlüssel recht exakt mit kleinem Speicher abschätzen, solange  $p[E_n]$  nicht zu klein ist. Beachte aber, dass wir nur die Anzahl, nicht aber die seltenen oder häufigen Schlüssel selbst in Erfahrung bringen können.

Wenn wir Permutationen zufällig auswürfeln, dann benötigt Algorithmus 4.25 den unrealistisch großen Speicherplatz  $\Omega(k \cdot \log_2(n!)) = \Omega(k \cdot n \cdot \log_2 n)$ . Stattdessen sollten Hashfunktionen aus wesentlich kleineren Hashklassen eingesetzt werden: Die linearen Permutationen  $a \cdot x + b \bmod N$  stellen sich in praktischen Anwendungen als gut heraus und benötigen nur Speicherplatz  $O(\log_2 N)$ .

## 4.4 Algorithmen für Zeitfenster: Zählen der letzten Einsen

Die Fenstergröße  $N$  sei gegeben. Wir betrachten einen Datenstrom und möchten die Anzahl „auffälliger“ Daten in einem Zeitfenster der Länge  $N$  zählen. Wir modellieren dieses Problem als das Zählen der Einsen unter den letzten  $N$  Bits in einem Datenstrom von Nullen und Einsen

Eine Lösung mit Speicherkomplexität  $N$  ist trivial, kostet aber zuviel schnellen Speicher. Wenn wir aber auf exakten oder  $\varepsilon$ -approximativen Lösungen mit kleinem  $\varepsilon$  beharren, dann ist eine große Speicherkomplexität erzwungen:

---

### Aufgabe 46

**Zeige**, dass jeder deterministische Algorithmus, der die Anzahl der Einsen im Zeitfenster der Größe  $N$  mit relativem Fehler  $\frac{1}{k}$  bestimmt, mindestens Speicherplatz  $\Omega(k \log_2^2 \frac{N}{k})$  benötigt.

---

Wir stellen einen Algorithmus vor, der mit Speicherkomplexität  $O(\frac{1}{\varepsilon} \cdot \log_2^2 N)$  arbeitet und die Anzahl „Geschätzt“ ausgibt, so dass  $|\text{Geschätzt} - \text{Exakt}| \leq 2\varepsilon \cdot \text{Exakt}$ .

Wir bauen eine Datenstruktur aus Körben  $K_1, \dots, K_m$ . Der Korb  $K_i$  speichert die Anzahl der Einsen in einem sich dynamisch ändernden Zeitintervall  $Z_i$ . Körbe haben die folgenden Eigenschaften:

- (1) Die Zeitintervalle  $Z_1, \dots, Z_m$  bilden eine disjunkte Zerlegung eines  $\{1, \dots, N\}$  umfassenden Zeitraums.
- (2) Die Zeitintervalle sind zeitlich geordnet ( $Z_1 < \dots < Z_m$ ), wobei  $Z_1$  den gegenwärtigen Zeitpunkt und  $Z_m$  den letzten interessanten Zeitpunkt  $N$  der Vergangenheit enthält.
- (3) Korb  $K_i$  speichert den Zeitstempel des Korbs (nämlich das Alter der jüngsten erfassten Eins im Zeitintervall  $Z_i$ ) und die Korbgröße  $G_i$  (d.h. die Anzahl der Einsen im Zeitintervall  $Z_i$ ).
- (4) Die Korbgrößen  $G_i$  sind stets Zweierpotenzen und die Körbe  $K_1 \dots, K_m$  besitzen aufsteigende Korbgrößen.
- (5) Sei  $k = \lceil \frac{1}{\varepsilon} \rceil$ . Wenn  $2^h$  die Korbgröße des letzten Korbs  $K_m$  ist, dann gibt es zu jedem  $h' < h$  mindestens  $\frac{k}{2}$  und höchstens  $\frac{k}{2} + 1$  Körbe der Größe  $2^{h'}$ .

Die Speicherplatzkomplexität der Korbdatenstruktur ist  $O(k \cdot \log_2^2 N) = O(\frac{1}{\varepsilon} \cdot \log_2^2 N)$ , denn nach Eigenschaft (5) gibt es insgesamt  $O(k \cdot \log_2 N)$  Körbe und jeder Korb speichert  $O(\log_2 N)$  Bits gemäß Eigenschaft (3).

Aufgrund der Zeitstempel können wir feststellen, ob ein Korb noch aktuell ist: Wenn der Zeitstempel des letzten Korbs größer als  $N$  ist, dann kann der Korb entleert und wiederverwandt werden. Beachte, dass alle Körbe bis auf den letzten Korb die exakte Anzahl der Einsen ihres Zeitintervalls in ihrer Korbgröße festhalten, ohne allerdings das Ende des Zeitintervalls zu kennen. Wenn der letzte Korb  $K_m$  noch aktuell ist, dann können wir nur sagen, dass

$$\sum_{i=1}^{m-1} G_i + 1 \leq \text{Exakt} \leq \sum_{i=1}^{m-1} G_i + G_m$$

gilt. Unser Algorithmus gibt die Schätzung  $\sum_{i=1}^{m-1} G_i + \frac{G_m}{2}$  ab. Um den relativen Fehler durch  $\varepsilon$  zu beschränken, genügt also die Forderung

$$\frac{G_m/2}{\sum_{i=1}^{m-1} G_i + 1} \leq 2 \cdot \varepsilon$$

und hierzu genügt wiederum der Nachweis von

$$\frac{(G_m - 1)/2}{\sum_{i=1}^{m-1} G_i + 1} \leq \varepsilon.$$

Diese Ungleichung wird durch Eigenschaft (5) erzwungen, denn wenn Korb  $K_l$  die Korbgröße  $2^r$  besitzt, dann gibt es mindestens  $\frac{k}{2}$  Körbe der Größen  $1, 2, 4, \dots, 2^{r-1}$  und damit ist sogar

$$\sum_{i=1}^{l-1} G_i + 1 \geq \frac{k}{2} \cdot \sum_{i=0}^{r-1} 2^i \geq \frac{k}{2} \cdot (2^r - 1) \geq \frac{1}{\varepsilon} \cdot \frac{G_l - 1}{2}$$

für jedes  $1 \leq l \leq m$ .

#### Algorithmus 4.27 Das Zählen von Einsen in einem Zeitfenster

- (1) Zwei Variablen „TOTAL“ und „LETZTER“ speichern die Gesamtgröße aller Körbe, bzw. den Index des letzten Korbs.
- (2) Wenn ein neues Bit ankommt, dann überprüfe, ob der letzte Korb noch aktuell ist. Ist dies nicht der Fall, dann wird der letzte Korb entfernt und die Variablen TOTAL und LETZTER werden aktualisiert.
- (3) Wenn das neue Bit eine Null ist, dann wird es ignoriert. Ansonsten wird ein neuer Korb mit Korbgröße 1 erzeugt und der Zeitpunkt wird festgehalten. TOTAL wird um Eins erhöht.
- (4) Wenn es jetzt  $\frac{k}{2} + 2$  Körbe der Korbgröße 1 gibt, dann fasse die beiden ältesten Körbe zu einem Korb der Größe 2 zusammen und bestimme den Zeitstempel des neuen Korbs. Da es nach Zusammenfassung möglicherweise zu viele Körbe der Größe 2 gibt, ist die Überprüfung von Eigenschaft (5) für Korbgröße 2 fortzusetzen.... Die Variable LETZTER ist zu aktualisieren.

**Beispiel 4.1** Es sei  $\frac{k}{2} = 1$  und die Korbgrößen seien  $(1, 1, 2, 4, 4, 8, 8, 16, 32)$ . Wenn eine neue Eins eintrifft, dann sind Körbe zusammenzufassen und wir erhalten  $(1, 2, 2, 4, 4, 8, 8, 16, 32)$ . Nach zwei weiteren Einsen und der entsprechenden Korbzusammenfassung ergibt sich dann der Vektor  $(1, 2, 4, 8, 16, 16, 32)$ .

Beachte, dass jedes neue Bit mit Hilfe einer geeigneten zusätzlichen Datenstruktur im worst-case  $O(\log_2 N)$  Operationen verursacht, während die amortisierte (oder durchschnittliche) Laufzeit sogar nur konstant ist. Wir fassen unsere Ergebnisse zusammen:

**Satz 4.28** Für jedes  $N$  wird die Anzahl der Einsen unter den letzten  $N$  Bits mit relativem Fehler höchstens  $\varepsilon$  bestimmt. Die worst-case Laufzeit pro Bit ist  $O(\log_2 N)$  und die amortisierte Laufzeit ist konstant. Die Speicherplatzkomplexität ist durch  $O(\frac{1}{\varepsilon} \cdot \log_2^2 N)$  beschränkt.

## 4.5 Histogramme

Der Datenstrom bestehe aus der Folge  $x_1, \dots, x_n$  von natürlichen Zahlen. Ein  $k$ -Histogramm  $(Z_1, \dots, Z_k)$  zerlegt den Zeitraum  $\{1, \dots, n\}$  in  $k$  Zeitintervalle  $Z_1, \dots, Z_k$  und gibt für jedes Zeitintervall  $Z_i$  einen konstanten Wert  $w_i$  an. Damit das Histogramm das Verhalten der Folge

möglichst gut wiedergibt, wird für jedes Zeitintervall  $Z_i$  eine „Stufenhöhe“  $w_i$  bestimmt, so dass die quadratische Abweichung, also

$$\text{abweichung}_i(w) = \sum_{j \in Z_i} (x_j - w_i)^2$$

minimal ist. Um optimale Stufenhöhen zu bestimmen, setzen wir die Ableitung der Funktionen „abweichung<sub>i</sub>“ auf Null und erhalten die Bedingungen

$$\sum_{j \in Z_i} -2 \cdot (x_j - w_i) = 0 \Rightarrow w_i = \frac{1}{|Z_i|} \sum_{j \in Z_i} x_j.$$

Also ist der „Erwartungswert“  $E(Z_i) = \frac{1}{|Z_i|} \sum_{j \in Z_i} x_j$  eine optimale Wahl der Stufenhöhe  $w_i$ . Natürlich möchten wir ein optimales  $k$ -Histogramm berechnen und haben deshalb noch die Wahl einer optimalen Zerlegung des Zeitraums  $\{1, \dots, n\}$  in  $k$  Zeitintervalle. Wir definieren

$$\text{Varianz}(Z) = \sum_{j \in Z} (x_j - E(Z))^2$$

als die „Varianz“ des Zeitintervalls  $Z$  und fordern, dass der Wert

$$W(Z_1, \dots, Z_k) = \sum_{i=1}^k \text{Varianz}(Z_i) \quad (4.2)$$

des  $k$ -Histogramms  $(Z_1, \dots, Z_k)$  kleinstmöglich ist. Mit anderen Worten, ein optimales  $k$ -Histogramm entspricht einer Stufenfunktion, die die Folge in der euklidischen Norm optimal approximiert.

Wir beschreiben zuerst einen dynamischen Programmier-Algorithmus, der ein optimales Histogramm berechnet. Unser Algorithmus bestimmt

$$\text{opt}_p(m) = \min\{ W(Z_1, \dots, Z_p) \mid (Z_1, \dots, Z_p) \text{ ist ein } p\text{-Histogramm für } x_1, \dots, x_m \}$$

für alle Kombinationen  $1 \leq p \leq k$  und  $1 \leq m \leq n$ . Die Bestimmung gelingt mit Hilfe der Rekursion

$$\begin{aligned} \text{opt}_1(m) &= \text{Varianz}(\{1, \dots, m\}), \\ \text{opt}_p(m) &= \min_{m^* < m} \{ \text{opt}_{p-1}(m^*) + \text{Varianz}(\{m^* + 1, \dots, m\}) \}. \end{aligned} \quad (4.3)$$

Um die Varianzen effizient zu berechnen, bestimmen wir in einer Vorberechnung zuerst alle Präfixsummen  $\sum_{i=1}^m x_i$  und  $\sum_{i=1}^m x_i^2$  und können dann die Varianz eines Zeitintervalls in konstant vielen Schritten evaluieren.

Wir haben insgesamt  $k \cdot n$  Teilprobleme zu lösen und die Lösung eines Teilproblems gelingt in Zeit  $O(n)$ . Die Laufzeit ist also durch  $O(k \cdot n^2)$  beschränkt. Schließlich ist die Speicherkomplexität durch  $O(n)$  beschränkt, denn es genügt, die Werte  $\text{opt}_p(1), \dots, \text{opt}_p(m)$  und  $\text{opt}_{p-1}(1), \dots, \text{opt}_{p-1}(m)$  abzuspeichern. Damit sind die benötigten Ressourcen aber jenseits von Gut und Böse und wir erlauben deshalb  $(1 + \varepsilon)$ -approximative Histogramme. Wir beginnen mit zwei einfachen Beobachtungen.

**Beobachtung 4.1** Wenn  $Z' \subseteq Z$  für Zeitintervalle  $Z'$  und  $Z$  gilt, dann ist  $\text{Varianz}(Z') \leq \text{Varianz}(Z)$ .

**Beweis:** Die Behauptung ist offensichtlich, denn

$$\text{Varianz}(Z) = \sum_{j \in Z} (x_j - E(Z))^2 \geq \sum_{j \in Z'} (x_j - E(Z))^2 \geq \sum_{j \in Z'} (x_j - E(Z'))^2 = \text{Varianz}(Z').$$

**Beobachtung 4.2** Wenn  $m_1 \leq m_2$ , dann ist  $\text{opt}_p(m_1) \leq \text{opt}_p(m_2)$ .

**Beweis:** Wenn  $(Z_1, \dots, Z_p)$  eine Zerlegung von  $\{1, \dots, m_2\}$  ist, dann induziert  $(Z_1, \dots, Z_p)$  auch eine (nicht zwangsläufig optimale) Zerlegung von  $\{1, \dots, m_1\}$ . Der Wert der induzierten Zerlegung liegt wegen der ersten Beobachtung unterhalb des Werts der ursprünglichen Zerlegung.  $\square$

Die Rekursionsgleichung (4.3) ist also zusammengesetzt aus den (in  $m^*$ ) *monoton steigenden*  $\text{opt}(m^*)$  Werten und den (ebenfalls in  $m^*$ ) *monoton fallenden*  $\text{Varianz}(\{m^* + 1, \dots, m\})$  Werten. Allerdings erlaubt diese Tatsache keine schnellere Bestimmung des Minimums in (4.3).

**Beispiel 4.2** Die Zahlen  $a_1, \dots, a_n$  seien nicht-negativ. Wir setzen  $f(i) = \sum_{j=1}^i a_j$  und  $g(i) = \sum_{j=i}^n a_j$ . Offensichtlich ist  $f$  monoton wachsend in  $i$  und  $g$  monoton fallend. Weiterhin ist  $f(i) + g(i) = a_i + \sum_{j=1}^n a_j$  und die Bestimmung des minimalen  $f(i) + g(i)$ -Werts ist äquivalent zur Bestimmung des minimalen  $a_i$ . Allerdings ist eine 2-approximative Lösung sehr einfach zu bestimmen, da stets  $\sum_{j=1}^n a_j \leq f(i) + g(i) \leq 2 \cdot \sum_{j=1}^n a_j$  gilt.

Unsere schnelle Approximation berechnet  $p$ -Histogramme für jedes  $p \leq k$ , indem (4.3) nur in wenigen, sorgfältig ausgewählten Punkten  $m^*$  minimiert wird.

#### Algorithmus 4.29 Schnelle Bestimmung guter $k$ -Histogramme mit wenig Speicher

(1) Bisher wurden die Zahlen  $x_1, \dots, x_n$  verarbeitet und die Zahl  $x_{n+1}$  sei gerade erschienen. Zu diesem Zeitpunkt sind unter Anderem für jedes  $p$  ( $1 \leq p \leq k$ ) gespeichert:

- die Folge  $1 = \text{links}_1^p < \text{links}_2^p < \dots < \text{links}_{I(p)}^p \leq n$  linker Endpunkte.
- für jeden rechten Endpunkt  $\text{rechts}_i^p = \text{links}_{i+1}^p - 1$  die Werte  $\text{approx}_p(\text{rechts}_i^p)$  von  $p$ -Histogrammen für  $\{1, \dots, \text{rechts}_i^p\}$  sowie die Summen  $\sum_{i=1}^{\text{rechts}_i^p} x_i$  und  $\sum_{i=1}^{\text{rechts}_i^p} x_i^2$ .  
/\* Der Algorithmus bestimmt eine Zerlegung in  $p$  Intervalle und vereinigt dazu einige Intervalle  $[\text{links}_i^p, \text{rechts}_i^p]$ . Der Wert der Zerlegung von  $\{1, \dots, m\}$  ist  $\text{approx}_p(m)$ . \*/

Für jedes  $i \leq I(p)$  fordern wir die Invariante

$$\text{approx}_p(\text{rechts}_i^p) \leq (1 + \delta) \cdot \text{approx}_p(\text{links}_i^p)$$

für  $\text{rechts}_i^p = \text{links}_{i+1}^p - 1$ .

/\* Die Invariante fordert insbesondere, dass die Varianz des Intervalls  $[\text{links}_i^p, \text{rechts}_i^p]$  nicht zu groß ist. Die Invariante garantiert eine gute Approximation, denn wir zeigen  $\text{approx}_p(n) \leq (1 + \delta)^{p-1} \cdot \text{opt}_p(n)$  in Lemma 4.30. \*/

- (2)  $p$ -Histogramme für  $\{1, \dots, n+1\}$  werden iterativ bestimmt. Für  $p = 1$  setzen wir  $\text{approx}_1(n+1) = \text{Varianz}(\{1, \dots, n+1\})$ : Approximation und Optimum fallen für  $p = 1$  zusammen. Im Schritt von  $p$  auf  $p+1$  setzen wir

$$\text{approx}_{p+1}(n+1) = \min_{i \in I(p)} \text{approx}_p(\text{rechts}_i^p) + \text{Varianz}(\{\text{links}_{i+1}^p, \dots, n+1\}),$$

Wenn  $\text{approx}_{p+1}(n+1) = \text{approx}_p(\text{rechts}_i^p) + \text{Varianz}(\{\text{links}_{i+1}^p, \dots, n+1\})$ , dann gibt der Algorithmus die Zerlegung  $Z_p(\text{rechts}_i^p) \cup [\text{rechts}_i^p + 1, n+1]$  aus, wobei  $Z_p(\text{rechts}_i^p)$  eine Intervallzerlegung von  $\{1, \dots, \text{rechts}_i^p\}$  in  $p$  Intervalle mit Wert  $\text{approx}_p(\text{rechts}_i^p)$  ist. Eine geeignete Datenstruktur muss die Bestimmung von  $Z_p(\text{rechts}_i^p)$  erleichtern.

/\* Wir bestimmen also ein bestes  $p+1$ -Histogramm, indem wir die gegebene Intervallzerlegung  $\{1, \dots, n\} = \bigcup_{i=1}^p [\text{links}_i^p, \text{rechts}_i^p]$  an allen rechten Endpunkten  $\text{rechts}_i^p$  aufspalten und ein jeweils bestes approximatives  $p$ -Histogramm bis zum Endpunkt  $\text{rechts}_i^p$  mit dem verbleibenden Intervall zusammensetzen. Wir wählen das Beste der untersuchten  $p+1$ -Histogramme als unsere approximative Lösung aus. \*/

Wenn  $\text{approx}_{p+1}(n+1) > (1 + \delta) \cdot \text{approx}_{p+1}(\text{links}_{I(p)}^p)$ , dann fügen wir das Intervall  $\{n+1\}$  hinzu, d.h. wir setzen  $I(p) = I(p) + 1$  und  $\text{links}_{I(p)} = n+1$ . Ansonsten bleibt die Folge linker Endpunkte unverändert.

Wir verifizieren zuerst, dass Algorithmus 4.29 gute  $p$ -Histogramme für jedes  $p \leq k$  berechnet.

**Lemma 4.30** Für jedes  $p \leq k$  gilt  $\text{approx}_p(n) \leq (1 + \delta)^{p-1} \cdot \text{opt}_p(n)$ .

**Beweis:** Wir führen eine Doppelinduktion, zuerst nach  $p$  und dann nach  $n$  durch. Für  $p = 1$  darf nur ein einziges Zeitfenster gewählt werden und die berechnete Lösung stimmt mit der optimalen Lösung überein. Wir betrachten als Nächstes den Fall von  $p+1$  Zeitfenstern.

Mit der Rekursionsgleichung (4.3) wissen wir, dass ein  $x$  mit

$$\text{opt}_{p+1}(n+1) = \text{opt}_p(x) + \text{Varianz}(\{x+1, \dots, n+1\})$$

existiert. Sei  $[\text{links}_i^p, \text{rechts}_i^p]$  das Intervall in dem  $x$  liegt. Da  $\text{opt}_p$  monoton wächst (Beobachtung 4.2), folgt

$$\text{opt}_p(\text{links}_i^p) \leq \text{opt}_p(x) \leq \text{opt}_p(\text{rechts}_i^p).$$

Weiterhin folgt aus der Invariante in Verbindung mit der Induktionsvoraussetzung

$$\begin{aligned} \text{approx}_p(\text{rechts}_i^p) &\leq (1 + \delta) \cdot \text{approx}_p(\text{links}_i^p) \\ &\leq (1 + \delta) \cdot (1 + \delta)^{p-1} \cdot \text{opt}_p(\text{links}_i^p) \leq (1 + \delta)^p \cdot \text{opt}_p(x) \end{aligned}$$

und, da Varianzen monoton fallen (Beobachtung 4.1), gilt

$$\begin{aligned} \text{opt}_{p+1}(n+1) &= \text{opt}_p(x) + \text{Varianz}(\{x+1, \dots, n+1\}) \\ &\geq (1 + \delta)^{-p} \cdot \text{approx}_p(\text{rechts}_i^p) + \text{Varianz}(\{x+1, \dots, n+1\}) \\ &\geq (1 + \delta)^{-p} \cdot \text{approx}_p(\text{rechts}_i^p) + \text{Varianz}(\{\text{rechts}_i^p + 1, \dots, n+1\}) \\ &\geq (1 + \delta)^{-p} \cdot \left( \text{approx}_p(\text{rechts}_i^p) + \text{Varianz}(\{\text{rechts}_i^p + 1, \dots, n+1\}) \right) \\ &\geq (1 + \delta)^{-p} \cdot \text{approx}_{p+1}(n+1). \end{aligned}$$

Und das war zu zeigen. □

Wir analysieren die benötigten Zeit und Platz-Ressourcen und bestimmen dazu zuerst die Anzahl linker Endpunkte. Wir fixieren  $p$  ( $1 \leq p \leq k$ ) und beobachten, dass Algorithmus 4.29 nur dann  $n + 1$  als neuen linken Endpunkt hinzufügt, wenn sich die approx-Werte des linken und des rechten Endpunkts des gegenwärtig letzten Intervalls um mehr als den Faktor  $1 + \delta$  unterscheiden.

Sei MAX die größte der  $n$  Zahlen  $x_1, \dots, x_n$ . Dann ist  $\text{approx}_p(\text{rechts}_i^p) \leq \sum_{j=1}^n x_j^2 \leq n \cdot \text{MAX}^2$  und damit werden höchstens

$$O\left(\log_{1+\delta}\left(n \cdot \text{MAX}^2\right)\right) = O\left(\frac{\log_2(n \cdot \text{MAX}^2)}{\log_2(1+\delta)}\right)$$

linke Endpunkte hinzugefügt. Wir wählen  $\delta$  so, dass  $(1+\delta)^k = 1+\varepsilon$  gilt. Wegen  $k \cdot \log_2(1+\delta) = \log_2(1+\varepsilon)$  erhalten wir insbesondere  $\log_2(1+\delta) = \log_2(1+\varepsilon)/k$ . Also ist die Anzahl linker Endpunkte für jedes  $p \leq k$  durch

$$O\left(\log_{1+\delta}\left(n \cdot \text{MAX}^2\right)\right) = O\left(\frac{k \cdot \log_2(n \cdot \text{MAX}^2)}{\log_2(1+\varepsilon)}\right) = O\left(\frac{k \cdot \log_2(n \cdot \text{MAX})}{\varepsilon}\right).$$

beschränkt. (Wir haben  $\ln(1+x) = \Theta(x)$  benutzt: Siehe Lemma 1.2.)

---

#### Aufgabe 47

Beschreibe eine Datenstruktur für Algorithmus 4.29. Laufzeit und Speicherplatz sollten durch  $O\left(\frac{k^2}{\varepsilon} \cdot \log_2(n \cdot \text{MAX})\right)$  beschränkt sein.

---

Wir können zusammenfassen:

**Satz 4.31** *Wenn die nte Zahl mit Approximationskonstante höchstens  $1 + \varepsilon$  zu verarbeiten ist, dann ist Speicherplatz und Laufzeit*

$$O\left(\frac{k^2}{\varepsilon} \cdot \log_2(n \cdot \text{MAX})\right)$$

*ausreichend, wobei  $\text{MAX} = \max\{x_1, \dots, x_n\}$ .*

Für kleine Werte von  $k$  genügt also logarithmischer Speicherplatz und logarithmische Laufzeit pro Zahl.

---

#### Aufgabe 48

Wir nennen eine Stichprobe der Größe  $k$  *zufällig und gleichverteilt* gezogen, wenn jede mögliche  $k$ -elementige Teilmenge aus den letzten  $m \geq k$  Punkten des Datenstroms die gleiche Wahrscheinlichkeit  $\binom{m}{k}^{-1}$  hat. Für  $t \geq m$  betrachten wir zufällig gezogene Stichproben  $S_t$  der Größe  $k$  auf dem Zeitfenster  $\{t - m + 1, \dots, t\}$  und nennen die Folge  $(S_t)_{t \geq m}$  *paarweise unabhängig*, wenn je zwei Stichproben für disjunkte Zeitfenster unabhängig sind.

Betrachte den folgenden Algorithmus **A** für den Datenstrom  $x_1, x_2, \dots$ :

- (1) Führe auf  $x_1, \dots, x_m$  Reservoir Sampling aus, um eine Stichprobe  $S_m$  der Größe  $k$  zu erhalten. Speichere die Datenpunkte der Stichprobe in aufsteigender (Alters-)Reihenfolge in einer Schlange ab.
- (2) Wiederhole für jedes  $t = m + 1, m + 2, \dots$ : Falls der älteste Datenpunkt der Schlange zum Zeitpunkt  $t - m$  in die Stichprobe aufgenommen wurde, entferne ihn und nimm  $x_t$  auf. Ansonsten ignoriere  $x_t$ .
- (a) Zeige, dass Algorithmus **A** eine zufällig und gleichverteilt gezogene Stichprobe auf den letzten  $m$  Datenpunkten liefert. **Hinweis:** Was passiert mit dem charakteristischen Vektor der Stichprobe  $S_m$  im Laufe der Zeit?
- (b) Zeige, dass die vom Algorithmus **A** gelieferte Stichproben  $(S_t)_{t \geq m}$  *nicht* paarweise unabhängig sind.

**Aufgabe 49**

Algorithmus **A** produziert eine periodische Stichprobe, die viele Anwendungen unmöglich macht. Wir führen deshalb einen neuen Algorithmus **B** ein, der eine Stichprobe der Größe 1 liefert. Sei  $x_1, x_2, \dots$  ein Datenstrom.

- (1)  $Q$  sei eine leere doppelt verkettete Liste, die in einem Listenelement sowohl den Index, als auch den Wert eines Datenpunkts speichern kann. Der Wert kann ungesetzt bleiben. Die Listenelemente haben also die Form  $(t, x_t)$  oder  $(t, *)$ . Die einelementige Stichprobe  $S$  initialisieren wir durch  $S = \emptyset$ .
- (2) Wiederhole für jeden Datenpunkt  $x_t$  für  $t = 1, 2, \dots$ :

*Schritt a:* Falls das *erste* Listenelement von  $Q$  verfällt, d.h. den Index  $t - m$  trägt, dann entferne dieses Element vom Anfang der Liste  $Q$ . Wenn das *neue erste* Listenelement nun den Wert  $x_j$  hat, dann setze  $S = \{x_j\}$ .

*Schritt b:* Wenn das *erste* Listenelement von  $Q$  nicht verfällt, dann wirf eine Münze mit Erfolgswahrscheinlichkeit  $p = 1/\min(t, m)$ . Bei einem Erfolg setze  $S = \{x_t\}$ , lösche  $Q$  komplett und füge den Index  $t$  als einziges Listenelement in  $Q$  ein. Bei Mißerfolg tue nichts.

*Schritt c:* Falls  $t$  nun als Index im *letzten* Listenelement von  $Q$  gespeichert ist, speichere dort zusätzlich den Wert  $x_t$ . Wähle dann zufällig gleichverteilt einen Index  $i$  aus der Menge  $\{t + 1, \dots, t + m\}$ . Füge diesen Index als neues Listenelement am Ende von  $Q$  ein.

- (a) Zeige: Zu jedem Zeitpunkt  $t \geq m$  (nach der Iteration für  $t$ ) gibt es ein  $r$  mit

$$Q[1] = (i_1, x_{i_1}), \dots, Q[r-1] = (i_{r-1}, x_{i_{r-1}}) \quad \text{und} \quad Q[r] = (i_r, *, )$$

wobei  $t - m + 1 \leq i_1 < i_2 < \dots < i_{r-1} \leq t < i_r$  und wobei  $x_{i_1}$  die gegenwärtige Stichprobe ist.

$Q$  besitzt also stets den gegenwärtigen Stichprobenschlüssel als erstes Listenelement. Nachfolgende Listenelemente sind nach Zeitpunkt aufsteigend angeordnet und spezifizieren den jeweils neuen Stichprobenschlüssel, wenn der gegenwärtige Stichprobenschlüssel verfällt.  $Q$  ist also eine Liste von Nachrückern.

- (b) Zeige, dass Algorithmus **B** eine zufällig und gleichverteilt gezogene Stichprobe auf den letzten  $m$  Datenpunkten liefert. Gilt dieselbe Aussage auch, wenn wir Schritt **b** auslassen?
- (c) Zeige oder widerlege, dass Algorithmus **B** eine Folge von paarweise unabhängigen Stichproben liefert.
- (d)  $|Q|$  sei die Länge der Liste  $Q$ . Zeige, dass  $\text{prob}[|Q| > \log m] \leq m^{-c}$  für jede Konstante  $c$  gilt.  
*Hinweis:* Zeige zuerst, dass  $\text{prob}[|Q| > r] \leq \binom{m}{r} m^{-r}$  gilt.
- (e) Zeige, dass die Ungleichung  $E[|Q|] \leq e$  für die erwartete Länge der Liste  $Q$  gilt.

*Hinweis:* Wenn eine Zufallsvariable  $X$  nur *natürliche* Zahlen als Werte annimmt, dann gilt  $E[X] = \sum_{i=0}^{\infty} \text{prob}[X > i]$ .

# Kapitel 5

## Queueing-Strategien

Queueing-Strategien haben einen großen Einfluss auf die Schnelligkeit der Beförderung. Wir nehmen an, dass jede Kante  $e$  eine Schlange  $s_e$  besitzt, in der die Pakete auf ihre Beförderung über die Kante  $e$  warten. Um die Notation zu vereinfachen, nehmen wir zusätzlich an, dass in jedem Schritt nur ein Paket über eine Kante befördert werden kann. Die Queueing-Strategie muss dann das zu transportierende Paket aus den wartenden Paketen auswählen.

Die FIFO-Strategie (First-In-First-Out) und ihre Varianten wird häufig angewandt. In Fair-Queueing wird für jede Kante  $e$  und für jeden Absender  $a$  eine Schlange  $s_{e,a}$  eingerichtet. Die Schlangen werden sodann für jede Kante  $e$  in Round-Robin Manier durchlaufen und das zu befördernde Paket der aktuellen Schlange wird nach dem FIFO-Prinzip bestimmt. Auf diese Art und Weise werden Absender mit geringem Paketverkehr nicht durch Absender mit großem Pakteverkehr benachteiligt. Gewichtetes Fair-Queueing ist eine zweite Variante; hier werden die Schlangen nicht nach Round-Robin, sondern gemäß den Gewichten der Schlangen abgearbeitet.

Wir betrachten nur Greedy-Strategien: Zu jedem Zeitpunkt wird ein Paket ausgewählt und weitergeleitet, wenn die jeweilige Schlange nichtleer ist. Neben der FIFO-Strategie sind vor Allem die folgenden Greedy-Strategien von Interesse:

- Farthest-To-Go (FTG): das Paket, das noch die meisten Kanten zu durchlaufen hat, wird gewählt.
- Nearest-To-Source (NTS): das Paket, das bisher die wenigsten Kanten durchlaufen hat, wird gewählt.
- Longest-In-System (LIS): das Paket, das die meiste Zeit im System verbracht hat, wird gewählt.
- Shortest-In-System (SIS): das Paket, das die kürzeste Zeit im System verbracht hat, wird gewählt.
- Nearest-To-Go (NTG): das Paket, das noch die wenigsten Kanten zu durchlaufen hat, wird gewählt.
- Farthest-From-Source (FFS): das Paket, das bisher die meisten Kanten durchlaufen hat, wird gewählt.

## 5.1 Stabilität

Wir haben verschiedene Auswertungsmöglichkeiten. Ein statistischer Ansatz aus der Theorie der Warteschlangen legt eine Emissionsverteilung zugrunde und misst das erwartete Verhalten und die Varianz. Natürlich ist hier nicht klar, welche Verteilung das tatsächliche Paketkommen am besten wiedergibt, und wir wählen stattdessen einen Worst-Case Ansatz:

Ein bösartiger Gegner versucht, die Anzahl der sich im System befindlichen Pakete „hochzuschaukeln“. Wir fragen, ob eine vorgegebene Queuing Strategie jeden Gegner in jedem Graphen in Schach halten kann.

**Definition 5.1**  $G$  sei ein gerichteter Graph.

- (a) Ein  $(r, b)$ -Gegner  $\mathcal{G}$  darf für jeden Knoten Pakete injizieren und deren Wege festlegen. Für jedes Zeitintervall  $I$  und für jede Kante  $e$  dürfen allerdings höchstens  $r \cdot |I| + b$  während  $I$  injizierten Pakete die Kante  $e$  in ihrem vorgeschriebenen Weg besitzen.  $r$  heißt die Injektionsrate und  $b$  die maximale Burstiness.

/\* Der Gegner erhält also die Möglichkeit, Wege für die von ihm injizierten Pakete vorzuschreiben, wobei aber keine Kante überladen werden darf. Damit wird ein Gegner im Allgemeinen über eine Kante  $e$  laufende Pakete in gleichmäßiger Geschwindigkeit injizieren. Allerdings sind bis zu  $b$  Ausnahmen erlaubt. \*/

- (b) Eine Queueing-Strategie ist auf  $G$  gegen einen Gegner  $\mathcal{G}$  stabil, wenn es eine Konstante  $C_G$  gibt, so dass zu jedem Zeitpunkt die Anzahl der im System verbleibenden Pakete durch  $C_G$  beschränkt ist, wenn die Pakete von  $\mathcal{G}$  in ein Anfangs leeres System injiziert wurden.

- (c) Eine Queueing-Strategie ist stabil, wenn die Strategie gegen jeden  $(r, b)$ -Gegner mit Injektionsrate  $r < 1$  auf jedem gerichteten Graphen stabil ist.

/\* Beachte, dass keine Queueing-Strategie gegen  $(r, b)$ -Gegner mit Injektionsrate  $r > 1$  stabil sein kann. \*/

Wir sind vor Allem an stabilen Queueing-Strategien interessiert. Eine stabile Queueing-Strategie muss für jeden Gegner und für jeden Graphen garantieren, dass kein Hochschaukeln erfolgt. Wir haben angenommen, dass eine Kante nur ein Paket transportieren kann und deshalb testet ein  $(r, b)$ -Gegner für  $r < 1$  die Queueing-Strategie nahe an der maximal tolerierbaren Auslastung: Für eine Injektionsrate von  $r > 1$  kann ein Hochschaukeln nicht verhindert werden.

Ein bösartiger  $(r, 0)$ -Gegner wählt die zum jeweiligen Zeitpunkt für die Queueing-Strategie schwierigsten Wege und hat zusätzlich die Möglichkeit „Bursts“ einzubauen, also zu einem Zeitpunkt sogar bis zu  $b$  Pakete zu injizieren, die (irgendwann) eine bestimmte Kante traversieren müssen.

Obwohl eine Injektionsrate  $r < 1$  unterhalb der maximalen Kapazität liegt, können Datenstaus aufgebaut werden: Wir setzen  $r = \frac{1}{2}$  und wählen  $N$  gerichtete Ketten  $K_{N-1}, \dots, K_0$  der Längen  $2 \cdot (N - 1), 2 \cdot (N - 2), \dots, 2, 0$ . Die Kettenköpfe besitzen den Knoten  $u$  als einzigen Nachfolger, der seinerseits nur den Nachfolger  $v$  besitzt. Wir provozieren einen Datenstau für die Kante  $(u, v)$  (für Injektionsrate  $r = \frac{1}{2}$ ), wenn wir im ersten Schritt ein Paket in das Kettenende der Kette  $K_{N-1}$ , im dritten Schritt ein Paket in das Kettenende der Kette  $K_{N-2}$

etc. injizieren. Nach Injektion eines Pakets in den einzigen Knoten von  $K_0$  wird eine Greedy Queueing-Strategie alle Pakete in den Knoten  $u$  gepumpt haben, und wir haben eine Schlange von  $N$  Paketen erzeugt, die alle auf ihren Transport über die Kante  $(u, v)$  warten.

Wenn eine Queueing-Strategie gegen einen Gegner stabil ist, dann besitzen alle Schlangen zu jedem Zeitpunkt eine Höchstzahl von Paketen. Bedeutet Stabilität aber auch, dass Pakete in beschränkter Zeit zugestellt werden? Gibt es einen direkten Zusammenhang zwischen maximaler Zustellzeit und der maximalen Größe von Schlangen?

Beide Fragen besitzen eine positive Antwort für  $r < 1$ , denn jede Schlange wird in einem genügend langen Zeitfenster mindestens einmal entleert und jedes wartende Paket kann somit die jeweilige Kante in diesem Zeitraum überqueren:

**Lemma 5.2 Die fundamentale Beobachtung**

*Wir fixieren eine Greedy Strategie. Sei  $e$  eine Kante eines vorgegebenen Graphen. Wenn es zum Zeitpunkt  $t$  genau  $k - 1$  Pakete im System gibt, die die Kante  $e$  in ihrer Wegbeschreibung besitzen, dann wird die Schlange von  $e$  irgendwann während der nächsten  $\frac{k+b}{1-r}$  Schritte leer sein.*

**Beweis:** Wir fixieren einen Zeitpunkt  $t$  und betrachten das Zeitfenster  $Z = [t + 1, t + \frac{k+b}{1-r}]$ . Wenn ein Paket  $\mathcal{P}$  die Schlange von  $e$  im Zeitfenster  $Z$  erreicht hat, dann war  $\mathcal{P}$  entweder zum Zeitpunkt  $t$  bereits im System oder aber ist eines von höchstens  $r \cdot \frac{k+b}{1-r} + b$  später injizierten Paketen. Also möchten insgesamt nur höchstens

$$k - 1 + r \cdot \frac{k + b}{1 - r} + b = \frac{(1 - r) \cdot (k + b - 1)}{1 - r} + r \cdot \frac{k + b}{1 - r} < \frac{k + b}{1 - r}$$

Pakete die Kante  $e$  im Zeitfenster  $Z$  durchlaufen und die Behauptung folgt, da wir nur Greedy Strategien betrachten.  $\square$

Die folgende Anwendung von Lemma 5.2 ist typisch:

Angenommen ein Paket  $\mathcal{P}$  wartet zum Zeitpunkt  $t$  in der Schlange einer Kante  $e$ . Wenn es zum Zeitpunkt  $t$  genau  $k - 1$  Pakete im System gibt, die auch die Kante  $e$  in ihrer Wegbeschreibung besitzen, dann wird  $\mathcal{P}$  die Kante  $e$  nach höchstens  $\frac{k+b}{1-r}$  Schritten überqueren.

Die nächste Aussage stellt einen Zusammenhang zwischen maximaler Schlangengröße und maximaler Zustellzeit her.

**Lemma 5.3**  *$Q$  sei eine Greedy-Queueing Strategie und  $\mathcal{G}$  sei ein  $(r, b)$ -Gegner mit  $r < 1$ ;  $G = (V, E)$  sei ein gerichteter Graph.*

- (a) *Es gelte  $k \cdot |E| > b$ . Wenn zu keinem Zeitpunkt eine Schlange mehr als  $k$  Pakete besitzt, dann wird jedes Paket, das mit einem Weg der Länge höchstens  $d$  injiziert wurde, nach höchstens  $\frac{2 \cdot k \cdot d \cdot |E|}{1-r}$  Schritten zugestellt.*
- (b) *Wenn jedes Paket nach höchstens  $s$  Schritten zugestellt wird, dann befinden sich zu jedem Zeitpunkt höchstens  $s$  Pakete in einer Schlange.*

**Beweis (a):** Nach Annahme besitzt jede Schlange höchstens  $k$  Pakete und damit befinden sich zu jedem Zeitpunkt höchstens  $k \cdot |E|$  Pakete im System. Wir betrachten eine beliebige Schlange  $S$  zu einem beliebigen Zeitpunkt  $T$  und wissen mit Lemma 5.2, dass  $S$  irgendwann während der nächsten  $\frac{k \cdot |E| + 1 + b}{1-r} \leq \frac{2 \cdot k \cdot |E|}{1-r}$  Schritte geleert wird.

Also überquert jedes Paket mindestens eine Kante innerhalb von  $\frac{2 \cdot k \cdot |E|}{1-r}$  Schritten und kann wie behauptet nach höchstens  $\frac{2 \cdot k \cdot d \cdot |E|}{1-r}$  Schritten zugestellt werden.

(b) Wenn sich zu irgendeinem Zeitpunkt mehr als  $s$  Pakete in einer Schlange  $S$  aufhalten, dann wird das niedrigst-rangige Paket in  $S$  für mehr als  $s$  Schritte aufgehalten.  $\square$

**Stabilität von Shortest-In-System:** Der Graph  $G = (V, E)$  ist ebenso vorgegeben wie der  $(r, b)$ -Gegner  $\mathcal{G}$  mit  $r < 1$ .

Um die maximale Verweildauer eines Pakets zu bestimmen, definieren wir die rekursive Folge  $(k_j)$  durch  $k_1 = b$  und  $k_{j+1} = \frac{k_j + b}{1-r}$ . Die Folge gibt die Maximalzahl  $k_j$  von rivalisierenden Paketen an, wenn bereits  $j$  Kanten traversiert wurden:

**Lemma 5.4** *Sei  $e$  eine beliebige Kante und sei  $\mathcal{P}$  ein beliebiges Paket, das  $e$  (irgendwann) durchlaufen möchte. Wenn  $\mathcal{P}$  die Schlange seiner  $j$ ten Kante erreicht, dann gibt es höchstens  $k_j - 1$  andere Pakete im System, die auch die Kante  $e$  in ihrer Wegbeschreibung besitzen und eine mindestens so hohe Priorität wie  $\mathcal{P}$  haben.*

**Beweis:** Sei  $e$  eine beliebige Kante und sei  $\mathcal{P}$  ein beliebiges Paket. Wir führen einen induktiven Beweis nach  $j$ . Wenn  $\mathcal{P}$  in seine erste Schlange eingefügt wird, dann haben nur die gleichzeitig mit  $\mathcal{P}$  injizierten Pakete eine gleichhohe Priorität. Aber in einem Schritt dürfen insgesamt höchstens  $\lfloor r + b \rfloor = b = k_1$  Pakete injiziert werden, die  $e$  traversieren möchten.

Für den Induktionsschritt betrachten wir zuerst die  $j$ te Kante  $e_j$  des Weges von  $\mathcal{P}$  und nehmen an, dass  $\mathcal{P}$  gerade die Schlange von  $e_j$  erreicht hat. Nach Induktionsannahme gibt es zu diesem Zeitpunkt höchstens  $k_j - 1$  Pakete, die  $e_j$  auch in ihrer Wegbeschreibung besitzen und eine mindestens so hohe Priorität wie  $\mathcal{P}$  haben. Wir wenden Lemma 5.2 an und erhalten, dass  $\mathcal{P}$  die  $j$ te Kante nach höchstens  $\frac{k_j + b}{1-r}$  Schritten überquert und damit die Schlange der Kante  $e_{j+1}$  erreicht.

In der Wartezeit können höchstens  $r \cdot \frac{k_j + b}{1-r} + b$  Pakete injiziert werden, die  $e$  als Kante in ihrer Wegbeschreibung besitzen. Wenn  $\mathcal{P}$  die Schlange von  $e_{j+1}$  erreicht, haben somit höchstens

$$k_j - 1 + r \cdot \frac{k_j + b}{1-r} + b = (1 - r + r) \cdot \frac{k_j + b}{1-r} - 1 = k_{j+1} - 1,$$

um  $e$  rivalisierende Pakete eine mindestens so hohe Priorität wie  $\mathcal{P}$ .  $\square$

**Satz 5.5** *SIS ist stabil. Wenn  $d$  die Länge eines längsten Weges ist, dann besitzt keine Schlange mehr als  $k_d$  Pakete und kein Paket verweilt für mehr als  $(d \cdot b + \sum_{i=1}^d k_i)/(1-r)$  Schritte im System.*

*Fazit: Wenn nur Wege logarithmischer Länge vorgegeben werden, dann verweilt jedes Paket nur für polynomiell viele Schritte im System.*

**Beweis:** Wenn zu irgendeinem Zeitpunkt mehr als  $k_d$  Pakete in der Schlange zu einer Kante  $e$  warten, dann betrachte ein Paket  $\mathcal{P}$ , das unter den wartenden Paketen die längste Zeit im System ist. Paket  $\mathcal{P}$  hat somit mehr als  $k_d - 1$  Rivalen, die  $e$  traversieren möchten und eine mindestens gleichhohe Priorität besitzen. Dies ist im Widerspruch zu Lemma 5.4, da  $k_d \geq k_j$  für alle  $j$ .

Weiterhin überquert ein Paket seine  $j$ te Kante nach höchstens  $(k_j + b)/(1-r)$  Schritten und damit folgt auch die Schranke für die Verweildauer.  $\square$

**Aufgabe 50**

Zeige, dass LIS stabil ist. Wenn  $d$  die Länge eines längsten Weges ist, dann besitzt jede Schlange höchstens  $\frac{b}{(1-r)^d} + b$  Pakete, und die Verweildauer eines Pakets ist höchstens  $\frac{b}{r \cdot (1-r)^d}$ .

**Stabilität von Farthest-To-Go:** Sei  $m$  die Kantenzahl von  $G$  und sei  $d$  die Länge eines längsten einfachen<sup>1</sup> Weges von  $G$ . Wir definieren die rekursive Folge  $(k_i)$  durch  $k_i = 0$  für  $i > d$  und  $k_i = m \cdot k_{i+1} + m \cdot b$  für  $1 \leq i \leq d$ .

**Satz 5.6** *FTG ist stabil. Es sind stets höchstens  $k_1$  Pakete im System, jede Schlange enthält höchstens  $k_2 + b$  Pakete und jedes Paket hat eine Verweildauer von höchstens*

$$(d \cdot (b + 1) + \sum_{i=2}^d k_i) / (1 - r).$$

**Beweis:** Wir zeigen durch eine Rückwärts-Induktion, dass zu jedem Zeitpunkt die Anzahl der Pakete, die noch mindestens  $i$  Kanten traversieren müssen, durch  $k_i$  beschränkt ist. Die Behauptung ist für  $i > d$  trivial, denn dann ist  $k_i = 0$ .

Sei  $e$  eine beliebige Kante. Wir definieren  $X_i(e, t)$  als

die Menge der Pakete in der Schlange von  $e$  zum Zeitpunkt  $t$ , die noch mindestens  $i$  Kanten traversieren müssen.

Desweiteren sei  $t^* < t$  der letzte Zeitpunkt, zu dem  $X_i(e, t^*)$  leer war. (Da das System anfänglich leer ist, existiert  $t^*$ .)

Wie gelangt ein Paket  $\mathcal{P}$  in die Menge  $X_i(e, t)$ ? Entweder wurde  $\mathcal{P}$  erst nach dem Zeitpunkt  $t^*$  injiziert oder  $\mathcal{P}$  musste zum Zeitpunkt  $t^*$  noch mindestens  $i + 1$  Kanten traversieren, denn es hatte die Schlange von  $e$  zum Zeitpunkt  $t^*$  noch nicht erreicht. Im ersten Fall ist  $\mathcal{P}$  eines von höchstens  $r \cdot (t - t^*) + b$  Paketen, denn höchstens  $r \cdot (t - t^*) + b$  im Zeitraum  $[t^* + 1, t]$  injizierte Pakete dürfen die Kante  $e$  in ihrer Wegbeschreibung besitzen. Im zweiten Fall ist  $\mathcal{P}$  nach Induktionsannahme eines von höchstens  $k_{i+1}$  Paketen.

Wir wenden die Farthest-To-Go Heuristik an und im Zeitfenster  $[t^* + 1, t]$  muss deshalb stets ein Paket mit noch mindestens  $i$  zu traversierenden Kanten die Schlange von  $e$  verlassen –denn  $t^*$  ist der letzte Zeitpunkt, zu dem  $X_i(e, t^*)$  leer war. Wir erhalten

$$\begin{aligned} |X_i(e, t)| &\leq k_{i+1} + r \cdot (t - t^*) + b - (t - t^*) \\ &= k_{i+1} - (1 - r) \cdot (t - t^*) + b \\ &\leq k_{i+1} + b. \end{aligned} \tag{5.1}$$

Also gibt es höchstens  $m \cdot k_{i+1} + m \cdot b = k_i$  Pakete, die noch mindestens  $i$  Kanten traversieren müssen und wir haben den Induktionsschritt abgeschlossen.

Die Größe der Schlange von  $e$  zum Zeitpunkt  $t$  stimmt mit  $|X_1(e, t)|$  überein. Da  $|X_1(e, t)| \leq k_2 + b$  aus (5.1) folgt, ist die maximale Größe einer Schlange durch  $k_2 + b$  beschränkt; als zusätzliche Konsequenz halten sich nie mehr als  $m \cdot (k_2 + b) = k_1$  Pakete im System auf.

Weiterhin folgt  $t - t^* \leq \frac{k_{i+1} + 1 + b}{1 - r}$  mit Lemma 5.2: Zum Zeitpunkt  $t^*$  war  $X_i(e, t^*)$  leer und alle zum Zeitpunkt  $t^*$  schon im System befindlichen Pakete aus  $X_i(e, t)$  mussten mindestens  $i + 1$  Kanten zurücklegen. Also wird jedes Paket mit  $i$  noch zu traversierenden Kanten nach

<sup>1</sup>Ein einfacher Weg traversiert jeden Knoten höchstens einmal.

höchstens  $\frac{k_{i+1}+1+b}{1-r}$  Schritten die nächste Kante traversiert haben, und die Verweildauer ist durch

$$\sum_{i=1}^d \frac{k_{i+1} + 1 + b}{1-r} = \frac{\sum_{i=2}^d k_i + d \cdot (b+1)}{1-r}$$

beschränkt, denn  $k_{d+1} = 0$ . □

#### Aufgabe 51

Gib einen geschlossenen Ausdruck für  $k_1, k_2$  und  $\sum_{i=2}^d k_i$  an.

#### Aufgabe 52

Wir betrachten die Nearest-To-Source Strategie, die stets die Pakete mit den wenigsten bereits durchquerten Kanten bevorzugt. Wir definieren eine Folge  $(l_i)$  mit  $l_0 = 0$  und  $l_i = m \cdot l_{i-1} + m \cdot b$  sonst.

- (a) Zeige :  $l_i = k_{d-i+1}$ .
- (b) Zeige, dass die Zahl der im System befindlichen Pakete, die bisher höchstens  $i$  Schritte zurückgelegt haben, zu jedem Zeitpunkt durch  $l_i$  beschränkt ist.
- (c) Leite Schranken für die maximale Paketzahl im System und für die maximale Transportzeit eines Paketes ab. Folgere daraus die Stabilität von NTS.

## 5.2 Instabile Queueing-Strategien

In den Konstruktionen benutzen wir das folgende Lemma, das eine beliebige Konfiguration als Anfangskonfiguration erlaubt.

**Lemma 5.7** Die Queueing-Strategie  $Q$  sei instabil gegen einen  $(r, b)$ -Gegner  $\mathcal{G}$  auf dem Graphen  $G$ , wenn mit einer (möglicherweise) nicht-leeren Konfiguration begonnen wird. Dann gibt es einen Graphen  $G^*$  und einen  $(r, b)$ -Gegner  $\mathcal{G}^*$ , so dass  $Q$  auch instabil auf  $G^*$  gegen  $\mathcal{G}^*$  ist, wenn mit der leeren Konfiguration begonnen wird.

**Beweis:** Wir kleben an jeden Knoten  $v$  von  $G$  einen Baum  $T_v$ , indem wir die Wurzel von  $T_v$  mit  $v$  identifizieren. Wenn Knoten  $v$  ursprünglich  $p_v$  Pakete erhalten hat, dann erhält die Wurzel genau  $p_v$  Kinder, die jeweils „Anführer“ einer genügend langen Kette sind. Der Baum  $T_v$  muss so gebaut sein, dass

- 1.) in jedem  $\frac{1}{r}$ ten der ersten  $\frac{1}{r} \cdot \sum_{v \in V} p_v$  Schritte genau ein Paket in ein Blatt injiziert wird,
- 2.) sich Pakete erst an der Wurzel ihres Baums treffen und
- 3.) die Pakete eines jeden Baums unter jeder Greedy-Strategie die Wurzel zum selben Zeitpunkt erreichen.

Sei  $t^*$  der Zeitpunkt, zu dem alle Pakete ihre Wurzel erreicht haben. Vom Zeitpunkt  $t^*$  ab, übernimmt der alte Gegner  $\mathcal{G}$  auf der jetzt erreichten Wunschkonfiguration. □

Wir weisen die Instabilität von FIFO bereits für ein System von vier Knoten  $a, b, c$  und  $d$  nach. Die vier Knoten bilden den Kreis  $a \rightarrow b \Rightarrow c \rightarrow d \Rightarrow a$ , wobei  $b$  über die zwei Kanten  $e_1 = (b, c)$  und  $e_2 = (b, c)$  mit seinem Nachfolger  $c$  und auch  $d$  über die zwei Kanten  $f_1 = (d, a)$  und  $f_2 = (d, a)$  mit seinem Nachfolger  $a$  verbunden ist.

**Satz 5.8** Für  $r \geq 0.85$  gibt es eine Anfangskonfiguration, auf der FIFO für den obigen Viererzyklus gegen einen  $(r, 0)$ -Gegner instabil ist.

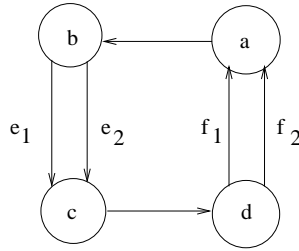


Abbildung 5.1: Ein schwieriger Graph für FIFO

**Beweis:** Wir konstruieren einen  $(r, 0)$ -Gegner in Phasen, so dass die Schlange von  $(a, b)$  zu Anfang der  $2 \cdot j$ ten Phase mindestens  $2 \cdot j$  Pakete enthält, während die Schlange von  $(c, d)$  am Ende der Phase mindestens  $2 \cdot j + 1$  Pakete enthält. Für ungerade Phasen  $2 \cdot j + 1$  wird die Konstruktion analog verlaufen, wobei wir aber mit mindestens  $2 \cdot j + 1$  Paketen in der Schlange von  $(c, d)$  beginnen und mit mindestens  $2 \cdot j + 2$  Paketen in der Schlange von  $(a, b)$  enden. Offensichtlich folgt hieraus die behauptete Instabilität.

Wir definieren den Gegner in Phase  $2 \cdot j$  unter der Voraussetzung, dass eine Menge  $W$  von  $n \geq 2 \cdot j$  Paketen in der Schlange von  $(a, b)$  wartet. Diese  $n$  Pakete werden dann später in ihrem Zielknoten  $b$  aus dem System genommen.

- 1.) Während der ersten  $n$  Schritte wird eine Menge  $X$  von  $r \cdot n$  Paketen mit der Wegbeschreibung  $(a, b), e_2, (c, d)$  in den Knoten  $a$  injiziert. Alle Pakete in  $X$  müssen nach Voraussetzung in der Schlange von  $(a, b)$  warten, da die  $n$  Pakete aus  $W$  nach der FIFO-Regel Vorfahrt haben.
- 2.) Während der nächsten  $r \cdot n$  Schritte wird eine Menge  $Y$  von  $r^2 \cdot n$  Paketen mit der Wegbeschreibung  $(a, b), e_1, (c, d)$  in den Knoten  $a$  injiziert. Diese Pakete müssen allerdings erst den vollständigen Transport der Pakete aus  $X$  über die Kante  $(a, b)$  abwarten.

Gleichzeitig wird der Weitertransport der jetzt in  $b$  ankommenden Pakete aus  $X$  über die Kante  $e_2$  aufgehalten, indem  $r^2 \cdot n$  Pakete mit Wegbeschreibung  $e_2$  in  $b$  injiziert werden. Diese Pakete vermischen sich mit den Paketen aus  $X$  und die Pakete aus  $X$  erhalten eine Dichte von  $\frac{1}{r+1}$  in der Schlange von  $e_2$ : In  $K$  Schritten werden  $r \cdot K$  Pakete neu injiziert,  $K$  Pakete aus  $X$  sind in der Schlange angekommen und insgesamt sind  $(r+1) \cdot K$  angekommen.

Dementsprechend kreuzen während dieser  $r \cdot n$  Schritte nur  $r \cdot n \cdot \frac{1}{r+1}$  Pakete aus  $X$  die Kante  $e_2$  und  $r \cdot n - \frac{r \cdot n}{r+1} = \frac{r^2 \cdot n}{r+1}$  Pakete aus  $X$  verbleiben in  $b$ . Alle Pakete aus  $Y$  befinden sich weiterhin im Knoten  $a$ , fließen aber in den nächsten Schritten ohne Verzug über die Kanten  $(a, b)$  und  $e_1$  in den Knoten  $c$ .

- 3.) Während der nächsten  $r^2 \cdot n$  Schritte bewegen sich die  $\frac{r^2 \cdot n}{r+1}$  Pakete aus  $X$  und die  $r^2 \cdot n$  Pakete aus  $Y$  über die Kanten  $e_2$  bzw.  $e_1$  und treffen sich im Knoten  $c$ . Gleichzeitig werden  $r^3 \cdot n$  Pakete mit Wegbeschreibung  $(c, d)$  im Knoten  $b$  injiziert.

Damit rivalisieren insgesamt  $r^2 \cdot n + \frac{r^2 \cdot n}{r+1}$  Pakete aus  $X \cup Y$  mit den neu injizierten  $r^3 \cdot n$  Paketen um die Kante  $(c, d)$  und nach diesen  $r^2 \cdot n$  Schritten enthält die Schlange von  $(c, d)$  mindestens  $r^3 \cdot n + \frac{r^2 \cdot n}{r+1}$  Pakete.

Wir beachten, dass  $r^3 \cdot n + \frac{r^2 \cdot n}{r+1} > n$ , denn  $r^3 \cdot (r+1) + r^2 > r+1$  für  $r \geq 0.85$ . Da die Konstruktion auch für ungerade Phasen analog verläuft, ist die Behauptung gezeigt.  $\square$

Es kann gezeigt werden, dass FIFO selbst für beliebig kleine Injektionsraten  $r > 0$  instabil ist [KMS]. Da FIFO von vielen Routern benutzt wird, kann damit ein „schleichender“ Denial-of-Service Angriff erfolgreich durchgeführt werden, solange der Angreifer die Wegbeschreibung der Pakete kontrolliert: Selbst nur langsam eingefügte Pakete können nicht schnell genug vom System befördert werden.

**Satz 5.9** *Nearest-To-Go ist für  $r > \frac{1}{\sqrt{2}}$  instabil gegen einen  $(r, 0)$ -Gegner.*

**Beweis:** Wir modifizieren das „FIFO-Netz“ geringfügig und arbeiten stattdessen mit einem Netz aus sechs Knoten.

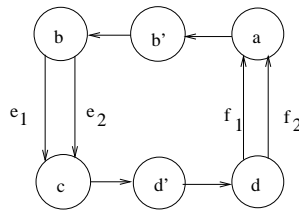


Abbildung 5.2: Ein schwieriger Graph für NTG

Wie in der Konstruktion für FIFO schaukeln wir das System langsam hoch. Wir machen zu einem beliebigen Zeitpunkt in unserer Konstruktion die folgende Annahme: Wenn wir keine weiteren Pakete injizieren, dann wird in den nächsten  $N$  Schritten jeweils ein Paket mit Ziel  $b'$  oder  $b$  die Kante  $(a, b')$  durchlaufen. Wir injizieren jetzt Pakete in zwei Phasen, um *mehr als  $N$  Pakete* mit Ziel  $d'$  oder  $d$  zu produzieren, die den Knoten  $d'$  noch nicht erreicht haben. **Phase 1** dauert  $N$  Schritte und wir injizieren in diesem Zeitraum eine Menge  $X$  von  $r \cdot N$  Paketen mit Wegbeschreibung  $(a, b')$ ,  $(b', b)$ ,  $e_1$ ,  $(c, d')$ . Am Ende dieser Phase bewegt Nearest-To-Go nur die  $N$  ursprünglichen Pakete, während alle Pakete aus  $X$  warten müssen.

**Phase 2** dauert  $r \cdot N$  Schritte. Wir injizieren zwei Paketmengen  $Y$  und  $Z$  von jeweils  $r^2 \cdot N$  Paketen. Die Pakete in  $Y$  haben den langen Weg  $(a, b')$ ,  $(b', b)$ ,  $e_2$ ,  $(c, d')$ ,  $(d', d)$  als Wegbeschreibung, während die Pakete in  $Z$  den sehr kurzen Weg  $e_1$  durchlaufen müssen.

Am Ende der zweiten Phase haben alle Pakete aus  $X$  den Knoten  $a$  verlassen, während alle Pakete aus  $Y$  in  $a$  warten müssen: Das Ziel  $d$  für die Pakete aus  $Y$  ist weiter entfernt als das Ziel  $d'$  für die Pakete aus  $X$ . Die Pakete aus  $X$  müssen aber die  $r^2 \cdot N$  Pakete aus  $Z$  an sich vorbeiziehen lassen und am Ende der zweiten Phase haben deshalb  $r^2 \cdot N$  Pakete aus  $X$  und alle  $r^2 \cdot N$  Pakete aus  $Y$  den Knoten  $d'$  noch nicht erreicht.

Wenn wir also keine weiteren Pakete injizieren, dann wird in den nächsten  $M \geq 2 \cdot r^2 \cdot N > N$  Schritten jeweils ein Paket mit Ziel  $d'$  oder  $d$  die Kante  $(c, d')$  durchlaufen. Wir treffen also eine im Vergleich zum Anfang der Konstruktion ähnliche Situation mit diesmal aber mehr als  $N$  involvierten Paketen an. Die Instabilität von Nearest-To-Go ist nachgewiesen, wenn wir unsere bisherige, im Knoten  $a$  beginnende Konstruktion jetzt im Knoten  $c$  entsprechend beginnen lassen.  $\square$

#### Aufgabe 53

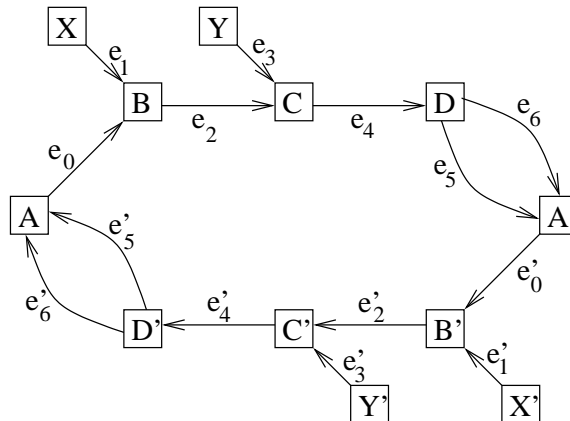
Zeige, dass auch Farthest-From-Source gegen einen  $(r, 0)$ -Gegner für  $r > \frac{1}{\sqrt{2}}$  instabil ist.

**Fazit:** Farthest-To-Go, Nearest-To-Source, Longest-In-System und Shortest-In-System sind stabil. Die populäre FIFO Strategie wie auch Nearest-To-Go und Farthest-From-Source sind hingegen instabil; Gleiches kann auch für die LIFO-Strategie gezeigt werden. Diese Strategien können also durch böse verteilt Pakete in die Knie gezwungen werden.

---

**Aufgabe 54**

Die LIFO-Strategie gibt stets dem Paket den Vorzug, welches die geringste Zeit wartet. Wir wollen zeigen, dass LIFO instabil ist, indem wir einen Graphen und eine Injektionsfolge angeben, die es uns erlaubt, *das System hochzuschaukeln*. Der folgende Graph bietet eine Möglichkeit dazu:



- (a) Nimm an, dieses Netzwerk wird mit LIFO betrieben und zu einem Zeitpunkt  $t$  gilt Folgendes: Würde ab Zeitpunkt  $t$  die Injektion neuer Pakete eingestellt, so würden noch für  $s$  Schritte jeweils ein Paket mit Ziel  $C$  und dem Restweg  $e_0, e_2$  die Kante  $e_0$  durchlaufen.  
Konstruiere eine  $(r, b)$  Injektionsfolge für die Schritte nach Zeitpunkt  $t$ , so dass schließlich zu einem Zeitpunkt  $t' > t$  gilt: Würde ab Zeitpunkt  $t'$  die Injektion neuer Pakete eingestellt, so würden noch für  $s'$  Schritte jeweils ein Paket mit Ziel  $C'$  und dem Restweg  $e'_0, e'_2$  die Kante  $e'_0$  durchlaufen, wobei  $s' > s$  gilt.
- (b) Warum folgt aus Teil *a* die Instabilität von LIFO?

**Hinweise:** Falls in einem Schritt mehrere Pakete in einem Knoten ankommen, so darf das weitergeleitete Paket unter diesen böse gewählt werden.

Es existiert eine Lösung, bei der zunächst  $rs$  Pakete in  $X$  injiziert werden und anschließend zeitgleich jeweils  $r^2s$  Pakete in  $Y$  und  $D$  injiziert werden.

---

Farthest-To-Go und Nearest-To-Source stechen hervor, da für sie sogar Stabilität für  $r = 1$  gezeigt werden kann [G]. Andererseits schneiden Farthest-To-Go und Nearest-To-Source mit maximaler Schlangengröße  $O(b \cdot |E|^{d-1}/(1-r))$  und Verweildauer  $O(b \cdot |E|^{d-1}/(1-r))$  zumindest in der Analyse leicht schlechter ab als Longest-In-System und Shortest-In-System mit maximaler Schlangengröße  $O(b/(1-r)^d)$  und Verweildauer  $O(b/(1-r)^d)$ , bzw.  $O(d \cdot b/(1-r)^d)$ , wenn  $|E| \gg 1/(1-r)$ .

Für alle betrachteten Strategien kann weder eine polynomielle maximale Schlangengröße noch eine polynomielle Verweildauer gezeigt werden. Wir erhalten nur dann polynomielle Beziehungen, wenn nur Wege logarithmischer Länge zu traversieren sind. Man beachte aber, dass sogar sublogarithmische Distanzen im Webgraphen zu erwarten sind und unsere Analysen zeigen, dass die vier obigen Queueing-Strategien im Anwendungsfall selbst unter worst-case „Attacken“ standhalten, wenn Kapazitäten nicht überschritten werden!

---

**Offenes Problem 2**

- (a) Besitzen Farthest-To-Go, Nearest-To-Source und Shortest-In-System polynomielle Verweildauer und polynomielle Schlangengrößen, wenn die Injektionsrate  $r$  hinreichend klein gewählt wird? Exponentiell große Verweildauer und Schlangengröße kann für hinreichend großes  $r < 1$  nachgewiesen werden [AAF].

Besitzt Longest-In-System eine polynomielle Verweildauer für alle  $r < 1$ ? Es ist bekannt, dass die Verweildauer exponentiell im Durchmesser sein kann.

(b) Ist Longest-In-System für  $r = 1$  stabil?

In [AAF] wird auch untersucht, ob es Graphen gibt, die unter jeder Queueing Strategie stabil sind und es wird gezeigt, dass der Ring für jede Knotenanzahl stabil ist. (Beachte, dass die anscheinend geringfügige Modifikation der beiden Doppelkanten im Viererzyklus zu einem instabilen Netz führt.) Die Klasse stabiler Graphen ist allerdings recht klein. Im wichtigen ungerichteten Fall gibt es sogar keine nicht-trivialen „stabilen Graphen“ und der Entwurf guter Queueing Strategien ist damit für jede interessante Netzarchitektur wichtig.

---

#### Aufgabe 55

Wir betrachten nur ungerichtete, zusammenhängende Graphen. Zeige, dass nur Graphen mit höchstens zwei Knoten unter jeder Queueing-Strategie stabil sind.

---

### 5.3 Queuegröße

Wir haben bisher stabile und instabile Queueing-Strategien kennengelernt, uns aber nicht mit der Frage nach der Queuegröße und der damit äquivalenten Frage nach der Transportzeit der einzelnen Strategien beschäftigt. Insbesondere muss uns interessieren, ob es Queueing-Strategien gibt, deren maximale Queuegröße oder maximale Transportzeit polynomiell im Durchmesser oder zumindest polynomiell in der Knotenanzahl ist. Wir werden hier eine leider negative Antwort für eine große Klasse von Strategien erhalten.

Allgemein kann man sich eine Queueing Strategie durch eine Prioritätsfunktion repräsentiert denken, die das vorhandene Wissen über das Netzwerk, den bisherigen Verlauf des Pakets und den noch zu durchlaufenden Weg auf eine Priorität des Pakets abbildet. Die Strategie gibt dann stets einem Paket mit maximaler Priorität den Vorzug. Klassen von Strategien entstehen dann natürlicherweise dadurch, dass man charakterisiert, welche Informationen die Prioritätsfunktion als Argument erhält. Wir betrachten die große Klasse der Strategien ohne Zeitnahme.

**Definition 5.10** Eine Queueing Strategie  $S_f$  arbeitet ohne Zeitnahme, wenn sie einem Paket  $p$  die Priorität  $f(G, P, a)$  zuweist, wobei  $G$  der Graph des Netzwerks,  $P$  der Weg von  $p$  und  $a$  die Zahl der bereits von  $p$  traversierten Kanten ist. Wir nennen Strategien, die ohne Zeitnahme arbeiten, auch WTS-Strategien (without time stamping.)

Prominente WTS-Strategien sind Nearest-To-Source (NTS), Farthest-From-Source (FFS), Nearest-To-Go (NTG), und Farthest-To-Go (FTG) mit den jeweiligen Prioritätsfunktionen  $f_{\text{NTS}}(G, P, a) = -a$ ,  $f_{\text{FFS}}(G, P, a) = a$ ,  $f_{\text{NTG}}(G, P, a) = a - |P|$  und  $f_{\text{FTG}}(G, P, a) = |P| - a$ . Wir zeigen, dass jede WTS-Strategie zu einer Queuegröße von  $2^{\Omega(\sqrt{n})}$  gezwungen werden kann, wobei  $n$  die Knotenzahl ist. Der Durchmesser  $d$  der Graphenfamilie ist dabei asymptotisch gleich  $\sqrt{n}$ , womit die Queuegröße  $2^{\Omega(d)}$  beträgt.

**Satz 5.11** [W] Es gibt eine Familie  $G_k$  von Graphen mit  $n = 2k^2 + 6$  Knoten und Durchmesser  $d = 4k$ , auf der jede WTS-Strategie mit Injektionsrate  $r > 0.5$  und Burstiness  $b > \frac{2rk}{2r-1}$  zu einer Queuegröße von  $2^{\Theta(k)}$  gezwungen werden kann. Damit ist die Queuegröße  $2^{\Theta(d)}$ , beziehungsweise  $2^{\Theta(\sqrt{n})}$ .

Dieses Resultat zeigt, dass Zeitmessung notwendig ist, wenn man an polynomiellen Queuegrößen interessiert ist, da die einzigen *sinnvollen* Größen, die in WTS-Strategien keinen

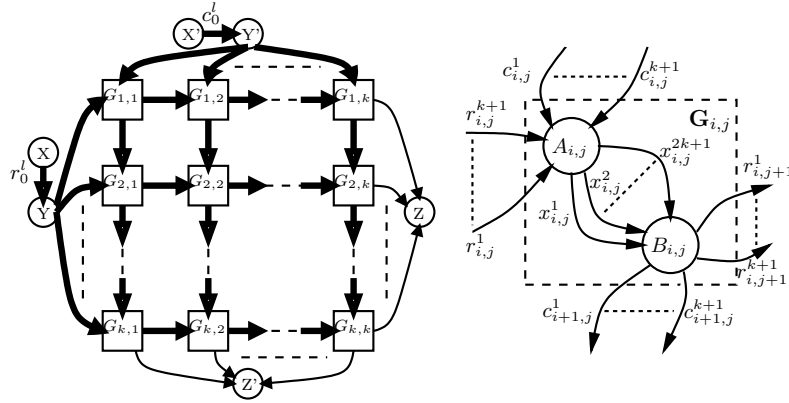


Abbildung 5.3: Graph  $G_k$  enthält  $k^2$  Gatter  $G_{i,j}$  sowie die zusätzlichen Knoten  $X, Y, Z, X', Y'$  und  $Z'$  angeordnet und verbunden wie links angedeutet. Ein dicker Pfeil steht hier für  $k + 1$  parallele Kanten in der jeweiligen Richtung. Jedes Gatter besteht aus zwei internen Knoten  $A_{i,j}$ , in dem alle eingehenden Kanten enden, und  $B_{i,j}$ , dem alle ausgehenden Kanten entspringen. Diese Knoten sind durch  $2k + 1$  gerichtete Kanten,  $x_{i,j}^1, \dots, x_{i,j}^{2k+1}$  von  $A_{i,j}$  nach  $B_{i,j}$  verbunden. Die eingehenden Zeilenkanten, die entweder aus  $G_{i,j-1}$  bzw. aus  $Y$  kommen, benennen wir  $r_{i,j}^l$  für  $1 \leq l \leq k + 1$ . Die eingehenden Spaltenkanten, die entweder aus  $G_{i-1,j}$  bzw.  $Y'$  kommen, benennen wir  $c_{i,j}^l$  für  $1 \leq l \leq k + 1$ . Die ausgehenden Zeilenkanten (bzw. Spaltenkanten) der letzten Spalte  $G_{i,k}$  (bzw. Zeile  $G_{k,j}$ ) nach  $Z$  [ $Z'$ ] heißen  $r_{i,k+1}$  [ $c_{k+1,j}$ ]. Schließlich nennen wir die  $k + 1$  Kanten von  $X$  nach  $Y$  (bzw.  $X'$  and  $Y'$ )  $r_0^l$  (bzw.  $c_0^l$ ) für  $1 \leq l \leq k + 1$ .

Eingang finden, Zeiten sind wie etwa die Lebenszeit eines Pakets – wie in LIS verwendet – oder die aktuelle Wartezeit eines Pakets – wie in FIFO verwendet.

**Beweis von Satz 5.11:**  $G_k = (V_k, E_k)$  besteht im Wesentlichen aus  $k^2$  Kopien des Gatters  $G_{ij}$  (vergl. Abb. 5.3).  $G_k$  hat  $2k^2 + 6$  Knoten und der Durchmesser beträgt  $4k$ .

Sei  $R_i$  die Menge der Wege von  $X$  zu  $Z$ , die ausschließlich die Gatter der Zeile  $i$  durchlaufen und sei  $C_j$  die Menge der Wege von  $X'$  zu  $Z'$ , die ausschließlich Gatter der  $j$ -ten Spalte durchlaufen. In unserer Konstruktion arbeiten wir ausschließlich mit diesen Zeilen- und Spaltenwegen.

Für jede der Kanten  $x_{i,j}^l$  bestimmen wir einen *dominanten* Weg  $W_{i,j}^l$ : Dieser Weg gehört zu  $R_i \cup C_j$ , benutzt  $x_{i,j}^l$  und erhält unter allen Paketwegen, die  $x_{i,j}^l$  durchlaufen, maximale Priorität in der Schlange von  $x_{i,j}^l$ . Damit besitzt ein Paket auf  $W_{i,j}^l$  also die Priorität

$$\max \left\{ \max_{P \in R_i} \{f(G_k, P, 2j) | P \text{ benutzt } x_{i,j}^l\}, \max_{P \in C_j} \{f(G_k, P, 2i) | P \text{ benutzt } x_{i,j}^l\} \right\},$$

da ein Paket auf einem Weg aus  $R_i$  (bzw.  $C_j$ ) genau  $2j$  Kanten (bzw.  $2i$  Kanten) durchlaufen hat, wenn es die Schlange von  $x_{i,j}^l$  erreicht. Falls  $W_{i,j}^l$  zu  $R_i$  gehört, nennen wir die Kante  $x_{i,j}^l$  *zeilendominiert* und andernfalls *spaltendominiert*. Darüberhinaus nennen wir Gatter  $G_{i,j}$  *zeilendominiert* (bzw. *spaltendominiert*) wenn mindestens  $k + 1$  seiner  $x_{i,j}^l$  Kanten zeilendominiert (bzw. spaltendominiert) sind. Also ist jedes Gatter entweder zeilen- oder spaltendominiert. Mindestens die Hälfte aller Gatter ist zeilendominiert oder mindestens die Hälfte aller Gatter ist spaltendominiert. Es gibt also eine Zeile, in der mindestens die Hälfte aller Gatter spal-

tendominiert sind, oder eine Spalte, in der mindestens die Hälfte aller Gatter zeilendominiert sind. Sei o.B.d.A. Zeile  $i_0$  eine solche Zeile.

Dann existieren  $q \geq \frac{k}{2}$  spaltendominierte Gatter in Zeile  $i_0$  und wir können ebenso o.B.d.A. annehmen, dass in spaltendominierten Gattern  $G_{i_0,j}$  die Kanten  $x_{i_0,j}^1, \dots, x_{i_0,j}^{k+1}$  spaltendominiert sind. Der folgende Algorithmus wählt die Wege aus, die bei der anschließenden Einfügung von Paketen benutzt werden.

1. Die Kantenmenge  $L$  enthalte die Kante  $r_{i_0,k+1}$ , die Kanten  $r_0^l$  sowie  $r_{i_0,j}^l$  und  $x_{i_0,j}^l$  für  $1 \leq j \leq k$  und  $1 \leq l \leq k+1$ . Wir nennen Kanten in  $L$  *legal* und nennen einen Weg *legal*, wenn er nur legale Kanten verwendet, in  $X$  beginnt und in  $Z$  endet.  
/\* Beachte dass legale Wege stets in Zeile  $i_0$  verlaufen. \*/
2. Sei  $z := q$ .  
/\*  $z$  durchläuft die spaltendominierten Gatter von rechts nach links. \*/
3. for  $j = k$  to 1 do
  - (a) Wähle  $e \in \{r_{i_0,j}^l | 1 \leq l \leq k+1\} \cap L$  beliebig.  
/\* Kante  $e$  ist also ein *legaler* Eingang in das Gatter  $G_{i_0,j}$ . \*/
  - (b) **Wenn**  $G_{i_0,j}$  spaltendominiert ist:
    - i. Sei  $S_z$  ein *legaler* Weg, der  $e$  benutzt, und unter allen *legalen* Wegen, die  $e$  benutzen, eine minimale Priorität in  $Q_e$  erhält:  $f(G_k, S_z, 2j-1)$  ist also *minimal*.
    - ii. Entferne die Kanten aus  $L$ , die  $S_z$  vor  $e$  durchläuft.
    - iii. Wähle eine beliebige Kante  $e' \in (\{x_{i_0,j}^l | 1 \leq l \leq k+1\} \setminus S_z)$ .  
/\* Kante  $e'$  ist also eine *legale* Kante, die von  $S_z$  nicht benutzt wird. \*/
    - iv. Sei  $D_z$  der *dominante* Weg von  $e'$ .  
/\* Die Kante  $e'$  ist spaltendominiert und  $D_z$  verläuft somit in Spalte  $j$ . Wir benutzen  $D_z$  später, um auf der Zeile  $i_0$  laufende Pakete aufzuhalten. \*/
    - v. Setze  $z := z - 1$ .
  - (c) **Sonst:** Wähle  $e' \in \{x_{i_0,j}^l | 1 \leq l \leq k+1\} \cap L$  beliebig.
  - (d) Entferne alle Kanten  $r_{i_0,j}^l \neq e$  und  $x_{i_0,j}^l \neq e'$  mit  $1 \leq l \leq k+1$  aus  $L$ .  
/\* Wir sagen, dass die Kanten  $e$  und  $e'$  fixiert sind. Später gewählte Wege  $S_{z'}$  (mit  $z' < z$ ) laufen somit über die bisher schon fixierten Kanten. Die Wege  $S_{z'}$  und  $S_z$  treffen sich aber wegen Schritt 3(b)ii zum ersten Mal in der Eingangskante  $e$ . \*/
4. Wähle einen *legalen* Weg  $S_0$  beliebig.  
/\*  $S_0$  ist bis auf die Mehrfachkante von  $X$  nach  $Y$  festgelegt. \*/

Man beachte, dass die Wahlen von  $e$  und  $e'$  jeweils wohldefiniert sind, da zu Beginn für jedes  $j$  mindestens  $k+1$  legale  $r_{i_0,j}^l$  und  $x_{i_0,j}^l$  Kanten existieren. Höchstens  $k-1$  dieser Kanten werden vor der Wahl von  $e$  und  $e'$  im Schritt 3(b)ii durch vorhergehende Schleifendurchläufe gelöscht.

Wir fügen nun in  $q$  Phasen Pakete ein, wobei die Phasen den spaltendominierten Gattern  $G_{i_0,j_1}, G_{i_0,j_2}, \dots, G_{i_0,j_q}$  der Zeile  $i_0$  entsprechen. Um den Prozess in Gang zu setzen, nutzen

wir einmalig die Burstiness aus und fügen im ersten Schritt  $b$  Pakete mit Wegbeschreibung  $S_0$  ins System ein. Diese Paketmenge nennen wir  $X_0$ .

**Phase  $t$**  beginnt, wenn das erste Paket von  $X_{t-1}$  die Queue der fixierten Kante  $r_{i_0, j_t}^l$  erreicht und dauert  $|X_{t-1}|$  Schritte. Wir fügen jetzt jeweils Pakete mit Rate  $r$  und Wegbeschreibungen  $S_t$  und  $D_t$  ein. Man beachte, dass  $S_t$  und  $D_t$  kantendisjunkt sind und diese Einfügungen belasten somit jede Kante mit Rate höchstens  $r$ :  $S_z$  und  $D_z$  haben lediglich das gemeinsame Gatter  $G_{i_0, j_t}$ ,  $S_z$  erreicht und verlässt es auf Zeilenkanten,  $D_z$  auf Spaltenkanten. Schließlich wird in Schritt 3(b)iii sichergestellt, dass  $S_z$  und  $D_z$  verschiedene  $x_{i_0, j}^l$  Kanten zur Durchquerung des Gatters benutzen.

In der Queue der fixierten Kante  $r_{i_0, j_t}^l$  treffen die Pakete aus  $S_t$  erstmalig auf die Pakete aus  $X_{t-1}$ . Ihre Priorität in dieser Queue ist nicht größer als die Priorität eines Paketes der Menge  $X_{t-1}$ , da jedes Paket aus  $X_{t-1}$  auf einem der Wege  $S_0, \dots, S_{t-1}$  unterwegs ist, und diese Wege alle legal waren, als  $S_t$  minimal unter den legalen Wegen gewählt wurde. Also kann keines der  $r|X_{t-1}|$  Pakete auf  $S_t$  das Gatter  $G_{i_0, j_t}$  in Phase  $t$  durchqueren, da die Eingangskante durchgehend von  $X_{t-1}$  Paketen benutzt wird.

Nach höchstens  $2k$  Schritten der Phase  $t$  erreicht das erste auf  $D_t$  eingefügte Paket das Gatter  $G_{i_0, j_t}$ . Von diesem Zeitpunkt an bis zum Ende der Phase (und das sind  $|X_{t-1}| - 2k$  Schritte) erreichen  $r \cdot (|X_{t-1}| - 2k)$  Pakete auf dem dominanten Spaltenweg das Gatter  $G_{i_0, j_t}$ .

Alle Pakete in  $X_{t-1}$  laufen über die fixierte Kante  $x_{i_0, j_t}^l$  und diese Pakete werden somit durch die Pakete des dominanten Spaltenweges aufgehalten. Daher schaffen es mindestens  $r \cdot (|X_{t-1}| - 2k)$  Pakete aus  $X_{t-1}$  nicht, das Gatter  $G_{i_0, j_t}$  in Phase  $t$  zu durchqueren.

Sei  $X_t$  die Vereinigung der Menge der aufgehaltenen  $X_{t-1}$ -Pakete und der neu auf  $S_t$  eingefügten Pakete. Dann ist

$$|X_t| \geq r(|X_{t-1}| - 2k) + rX_{t-1} = 2r|X_{t-1}| - 2rk.$$

Für  $r > \frac{1}{2}$  und hinreichend großes<sup>2</sup>  $b$  ist jede Menge  $X_t$  um (fast) den multiplikativen Faktor  $2r$  größer als  $X_{t-1}$ . Da  $q = \Theta(k) = \Theta(d)$  gilt, hat die letzte Menge  $X_q$  die Größe  $2^{\Theta(k)} = 2^{\Theta(d)} = 2^{\Theta(\sqrt{n})}$ , was zu zeigen war.  $\square$

## 5.4 Eine randomisierte Queueing-Strategie

Wir schließen mit einer randomisierten Queueing-Strategie, die hochwahrscheinlich polynomielle Verweildauer und polynomielle Schlangengröße besitzt. Wir analysieren dazu zunächst eine randomisierte Strategie für das *statische* Routing Problem. Beim statischen Routing Problem sind die zu transportierenden Pakete von Beginn an im System. Es werden keine weiteren Pakete zur Laufzeit hinzugefügt.

Es sei auch hier  $d$  die maximale Länge eines einfachen Weges in  $G$  und  $m$  die Zahl der Kanten. Sei weiter  $c$  das maximale Vorkommen einer Kante in den Wegen der Pakete, also die größte Zahl von Paketen, die über eine Kante transportiert werden sollen. Damit sind  $c$  und  $d$  triviale untere Schranke für die Laufzeit. Unsere statische Queueing Strategie wird mit hoher Wahrscheinlichkeit eine Laufzeit von

$$(1 + \varepsilon) \cdot c + O(d \cdot \log(mcd))$$

für beliebiges vorgegebenes  $\varepsilon$  erreichen.

<sup>2</sup>Die Bedingung  $|X_1| \geq 2r \cdot |X_0| - 2rk > |X_0|$  ist äquivalent zu  $|X_0| > \frac{2rk}{2r-1}$  und da,  $|X_0| = b$ , ist  $b > \frac{2rk}{2r-1}$  ausreichend.

**Algorithmus 5.12** Eine randomisierte Queueing-Strategie für statisches Routing

- (1) Wähle für jedes Paket unabhängig und gleichverteilt eine *Startverzögerung* aus der Menge  $\{1, \dots, \frac{\alpha c}{\log(mcd)}\}$ , wobei wir  $\alpha$  in der Analyse passend bestimmen werden. Setze  $i = 1$ .
- (2) Wiederhole, solange nicht zugestellte Pakete existieren:
  - (a) Bewege zum Zeitpunkt  $i$  alle Pakete mit Startverzögerung höchstens  $i$  um eine Kante entlang ihres Weges weiter. (Wann immer mehrere Pakete dieselbe Kante durchlaufen wollen, wird dies mehr als einen Zeitschritt erfordern.) Pakete mit höherer Startverzögerung werden nicht bewegt.
  - (b) Erhöhe  $i$  um 1.

Wir bestimmen zunächst die Zahl der Iterationen von Schleife (2). Ein Paket wartet höchstens  $\frac{\alpha c}{\log(mcd)}$  Iterationen und kommt dann in jeder Iteration dem Ziel um eine Kante näher. Nach  $d + \frac{\alpha c}{\log(mcd)}$  Iterationen sind also alle Pakete zugestellt.

Kritisch für die Laufzeit ist die Frage, wieviele Zeitschritte in Schritt (2a) nötig sind. Dafür müssen wir untersuchen, wieviele Pakete in derselben Iteration  $i$  dieselbe Kante durchlaufen möchten. Die Zufallsvariable  $N_{e,i}$  bezeichnet die Zahl der Pakete, die in Iteration  $i$  durch Kante  $e$  laufen möchten. Desweiteren ist  $I_{p,e,i}$  eine Indikatorvariable, die angibt, ob Paket  $p$  in Iteration  $i$  über Kante  $e$  laufen möchte. Es ist  $N_{e,i} = \sum_p I_{p,e,i}$  und wir beschränken den Erwartungswert:

$$\begin{aligned} E[N_{e,i}] &= E\left[\sum_p I_{p,e,i}\right] = \sum_p E[I_{p,e,i}] \\ &= \sum_p \text{prob}[\text{Paket } p \text{ will in Iteration } i \text{ die Kante } e \text{ durchlaufen}] \\ &\leq c \cdot \frac{\log(mcd)}{\alpha c} = \frac{\log(mcd)}{\alpha} \end{aligned}$$

Die Abschätzung ist richtig, da höchstens  $c$  Pakete überhaupt Kante  $e$  auf ihrem Weg haben und von den möglichen Startverzögerungen höchstens eine dazu führt, dass  $p$  genau in Iteration  $i$  über die Kante  $e$  laufen möchte.

Wir benutzen die Chernoff Schranke, um die Wahrscheinlichkeit von zu langen Iterationen zu beschränken. (Die Chernoff-Schranke ist anwendbar, da die Pakete ihre Startverzögerung unabhängig voneinander auswürfeln.)

$$\begin{aligned} \text{prob}\left[N_{e,i} > (1 + \varepsilon) \cdot \frac{\log(mcd)}{\alpha}\right] &\leq e^{-\frac{\log(mcd)}{\alpha} \cdot \frac{\varepsilon^2}{3}} \\ &= (mcd)^{-\varepsilon^2/(3\alpha)} \end{aligned}$$

Die Wahrscheinlichkeit, dass irgendeine Iteration der Schleife (2) länger als  $(1 + \varepsilon) \cdot \frac{\log(mcd)}{\alpha}$  dauert, beschränken wir, indem wir die gerade gefundene Wahrscheinlichkeit mit der Zahl  $m$  der Kanten und der maximalen Iterationszahl  $d + \frac{\alpha c}{\log(mcd)}$  multiplizieren.

$$\text{prob}[\text{irgendeine Iteration dauert länger als } (1 + \varepsilon) \cdot \frac{\log(mcd)}{\alpha}] \leq (mcd)^{\frac{-\varepsilon^2}{3\alpha}} \cdot m \cdot \left(d + \frac{\alpha c}{\log(mcd)}\right)$$

Wählen wir  $\alpha \leq \frac{c-1}{c} \cdot d \cdot \log(mcd)$ , so ist  $d + \frac{\alpha c}{\log(mcd)} \leq c \cdot d$ , und wir erhalten

$$\text{prob}[\text{irgendeine Iteration dauert länger als } (1 + \varepsilon) \cdot \frac{\log(mcd)}{\alpha}] \leq (mcd)^{\frac{-\varepsilon^2}{3\alpha} + 1}.$$

Zu jedem gegebenen  $\varepsilon$  können wir  $\alpha \leq \frac{\varepsilon^2}{6}$  wählen und damit ist die Wahrscheinlichkeit einer zu langen Iteration durch  $(mcd)^{-1}$  beschränkt. Multiplizieren wir die obere Schranke der Iterationsdauer mit der oberen Schranke  $d + \frac{\alpha c}{\log(mcd)}$  der Iterationsanzahl, so erhalten wir:

**Lemma 5.13** *Algorithmus 5.12 löst das statische Routingproblem mit Wahrscheinlichkeit  $1 - (mcd)^{-1}$  in Zeit*

$$(1 + \varepsilon) \cdot c + \frac{1 + \varepsilon}{\alpha} \cdot (d \cdot \log mcd) = (1 + \varepsilon) \cdot c + O(d \cdot \log mcd)$$

für jedes  $\varepsilon > 0$ .

Wir kommen zurück zu unserem eigentlichen Problem, dem dynamischen Routing. Wir wollen hierfür den folgenden Algorithmus analysieren. Den Parameter  $T$  legen wir in der Analyse geeignet fest.

**Algorithmus 5.14 Eine randomisierte Queueing-Strategie.**

- (1) Der Algorithmus arbeitet mit den Zeitintervallen  $Z_i = \{iT + 1, \dots, (i + 1)T\}$  der Länge  $T$ . Alle Pakete, die im Zeitintervall  $Z_i$  in das System gelangen, werden in eine Menge  $X_i$  aufgenommen. Wir sagen, dass  $X_i$  zum Zeitpunkt  $(i + 1)T$ , also nach Injektion aller Pakete im Zeitintervall  $Z_i$ , *aktiv* wird.  $X_i$  wird deaktiviert, wenn sämtliche Pakete in  $Z_i$  zugestellt wurden.
- (2) Wiederhole:
  - (a) Bestimme das kleinste  $k$ , so dass  $X_k$  aktiv ist.
  - (b) Führe auf den Paketen aus  $X_k$  den statischen Algorithmus aus. Nach seiner Beendigung sind alle Pakete aus  $X_k$  zugestellt und  $X_k$  wird deaktiviert.

Wir sollten  $T$  also so wählen, dass der Algorithmus, *wenn alles normal läuft*, eine Menge  $X_i$  zugestellt hat, bevor die nächste Menge  $X_{i+1}$  aktiv wird. Andernfalls besteht die Gefahr, dass wir uns lange Warteschlangen von aktiven Mengen einhandeln.

Wir sagen, dass der statische Algorithmus auf  $X_i$  erfolgreich ist, wenn alle Pakete aus  $X_i$  in Zeit

$$(1 + \varepsilon) \cdot (rT + b) + \frac{1 + \varepsilon}{\alpha} \cdot d \cdot \log(md(rT + b))$$

zugestellt werden. Da wir einen  $(r, b)$  beschränkten Gegner annehmen, wird  $X_i$  für jede Kante  $e$  höchstens  $rT + b$  Pakete besitzen, die  $e$  durchlaufen möchten. Wenn wir also  $c = rT + b$  setzen, dann ist der statische Algorithmus gemäß Lemma 5.13 auf  $X_i$  mit Wahrscheinlichkeit  $1 - (mcd)^{-1}$  erfolgreich.

Da die Fehlerwahrscheinlichkeit aber nur invers polynomiell ist, richten wir einen zusätzlichen Faktor von  $1 + \varepsilon$  als Fehlertoleranz ein. Wir setzen also  $T$  kleinstmöglich, so dass

$$T \geq (1 + \varepsilon)^2 \cdot \left( c + \frac{d}{\alpha} \cdot \log(mcd) \right).$$

Beachte, dass  $T$  auf der rechten Seite nur linear (in  $c = rT + b$ ) bzw. logarithmisch eingeht, so dass wir keine Probleme haben werden, ein solches  $T$  zu finden.

---

**Aufgabe 56**

Zeige  $T = \Theta\left(\frac{d \log(m)}{1-r \cdot (1+\varepsilon)^2}\right)$ .

---

Im Erfolgsfall schafft unser Algorithmus also eine Menge  $X_i$  sogar mit Zeitreserve, bevor die nächste Menge aktiv wird: Von der zur Verfügung stehenden Zeit  $T$  wird dann nur die Zeit  $T/(1+\varepsilon)$  verbraucht und damit kann der eventuelle Rückstand um mindestens  $T - \frac{T}{1+\varepsilon} = \frac{\varepsilon}{1+\varepsilon}T$  wieder abgetragen werden.

Sei  $\tau_i$  die Zeitverzögerung, nach der der statische Algorithmus die Verarbeitung von  $X_i$  übernimmt. Ist  $\tau_i > 0$ , so wird die Abarbeitung von  $X_i$  im Erfolgsfall den Rückstand additiv um  $\min\left\{T \cdot \frac{\varepsilon}{1+\varepsilon}, \tau_i\right\}$  verkleinern. Im Misserfolgsfall schätzen wir die zusätzliche Verzögerung pessimistisch durch  $d(rT + b) = dc$  ab: Die höchstens  $r \cdot T + b$  injizierten Pakete können nacheinander in Zeit höchstens  $d$  an ihr Ziel gebracht werden.

Wir nehmen an, dass  $c \geq 1$ ,  $m, d \geq 2$  und dass  $T$  so groß gewählt ist, dass  $\frac{\varepsilon}{1+\varepsilon} \cdot T \geq 4$ . Wenn  $\tau_i \geq T \cdot \frac{\varepsilon}{1+\varepsilon}$ , dann gilt also für die erwartete Veränderung im Hinblick auf die Planerfüllung

$$\begin{aligned} E[\tau_{i+1} - \tau_i] &\leq -\frac{\varepsilon}{1+\varepsilon} \cdot T \cdot \left(1 - \frac{1}{c \cdot d \cdot m}\right) + \frac{1}{c \cdot d \cdot m} \cdot c \cdot d \\ &\leq -\frac{\varepsilon}{1+\varepsilon} \cdot T \cdot \left(1 - \frac{1}{c \cdot d \cdot m}\right) + \frac{1}{m} \\ &\leq -4 \cdot \frac{3}{4} + \frac{1}{2} = -2.5. \end{aligned}$$

Die Folge  $\tau_1, \tau_2, \dots$  zeigt also das folgende Verhalten:

- (1)  $\tau_{i+1} - \tau_i \leq c \cdot d$  und
- (2)  $E[\tau_{i+1} - \tau_i] \leq -2.5$ , falls  $\tau_i \geq \frac{\varepsilon}{1+\varepsilon} \cdot T$ .

Unter diesen Bedingungen kann gezeigt werden [H], dass für jedes  $i$

$$\text{prob}[\tau_i \geq \alpha \cdot c \cdot d] \leq e^{-\Omega(\alpha)}$$

gilt.

**Satz 5.15** *Es gelte  $T = \Theta\left(\frac{d \log(m)}{1-r}\right)$  und  $c = r \cdot T + b$ . Dann gilt für jede Menge  $X_i$  mit Wahrscheinlichkeit mindestens  $1 - e^{-\Omega(\alpha)}$ , dass alle Pakete aus  $X_i$  nach höchstens*

$$2 \cdot T + O(c \cdot d) = O\left(\frac{d^2 \cdot \log_2 m}{1-r}\right)$$

*Schritten zugestellt werden. Also ist Algorithmus 5.14 stabil und besitzt hochwahrscheinlich eine polynomielle Verweildauer.*

# Kapitel 6

## Erasure Codes

In [YKT] wurde über den Zeitraum einer Stunde der Versand einer langen Datei an 11 Empfänger untersucht und eine durchschnittliche Verlustrate pro Empfänger von ungefähr 9% festgestellt. Auf der Seite des Senders mussten 46% aller Pakete an mindestens einen Empfänger mehrfach versandt werden. Der Paketverlust stellt somit ein ernstzunehmendes Problem dar, das selbst einen leistungsstarken Sender bei großen Multicast-Aufgaben (Versand langer Programmcodes oder Videos hoher Qualität an mehrere hunderttausende Empfänger) auf Grund der eintreffenden Anforderungen nach Neuversand von Paketen in die Knie zu zwingen droht. Multicasts, die zeit-nahe erledigt werden müssen, drohen damit zu scheitern.

Um den Paketverlust zu kompensieren liegt es natürlich nahe, die zu versendende Information *redundant* zu kodieren. Die Zielvorgaben sollten sein:

- (1) Rekonstruktion trotz hohen Paketverlusten. Wir geben keine Worst-Case Garantie, sondern garantieren Rekonstruktion nur mit hoher Wahrscheinlichkeit.
- (2) Superschnelle Kodierungs- und Dekodierungsalgorithmen, um die hohen Bandbreiten von Hochleistungskanälen (Glasfaser, ATM oder Gigabit Ethernet) ausnutzen zu können.

### Beispiel 6.1 Automobile Data Broadcasting

Ein Schlüsselproblem im Data-Broadcast für Autos ist die Anforderung verlässlicher Kommunikation mit einer Vielzahl von Empfängern über unsichere Kanäle (wechselnde Landschaften, Tunnel). Da eine Rückverbindung zum Empfänger nicht besteht, ist ein nicht-kompensierbarer Paket-Verlust äußerst schwerwiegend und redundante Kodierung ist erforderlich.

Wir beschreiben zuerst das formale Modell eines *Kanals  $K$  mit Verlust* (erasure-channel oder auch lossy channel genannt): Für ein Alphabet  $\Sigma$  werden Nachrichten der Form

$$(a_1, 1) \cdot (a_2, 2) \cdots (a_i, i) \cdots (a_n, n)$$

(mit  $a_1, \dots, a_n \in \Sigma$ ) über den Kanal  $K$  verschickt. (Das Alphabet  $\Sigma$  kann zum Beispiel aus allen Bitfolgen einer bestimmten Höchstlänge bestehen.)  $K$  löscht ein Paket  $(a_i, i)$  mit Wahrscheinlichkeit  $\rho$ , wobei die Wahrscheinlichkeit des Löschen eines Pakets nicht von den anderen Paketen abhängt. Beachte, dass jedes Paket  $(a_i, i)$  einen Zeitstempel, nämlich den Zeitpunkt  $i$ , enthält: Der Empfänger weiß also, welche Pakete verloren gegangen sind.

**Bemerkung 6.1** Auf den ersten Blick ist der Erasure-Kanal kein realistisches Modell für das Beispiel des Internets. Dort wird man erwarten, dass die Löschwahrscheinlichkeiten zeitlich benachbarter Pakete stark korreliert sind. Eine solche Korrelation können wir aber auflösen, wenn wir „gegen den Kanal randomisieren“, also die Pakete vor dem Versand zufällig permutieren.

Wir beschreiben die Tornado Codes, die selbst hohe Verlustraten tolerieren und zusätzlich noch superschnelle Kodier- und Dekodieralgorithmen anbieten. Um die Beschreibung zu vereinfachen, nehmen wir an, dass jedes Packet nur aus einem Bit und der Paketnummer besteht. Der Tornado Code wird durch eine Folge von „dünn-besetzten“ bipartiten Graphen  $B_i = (U_i, V_i, E_i)$  (mit  $|V_i| = \beta \cdot |U_i|$  für  $\beta < 1$  und  $|E_i| = O(|U_i|)$ ) definiert.

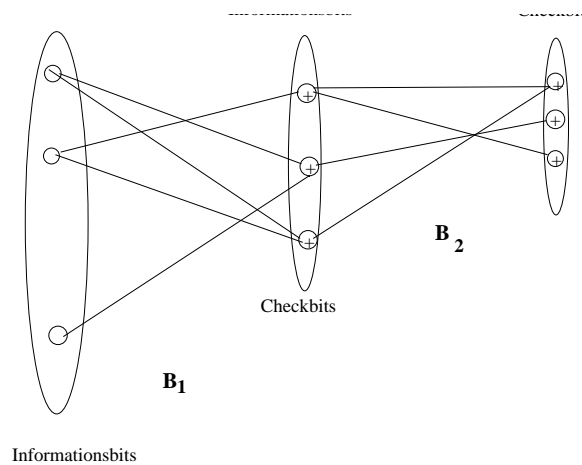


Abbildung 6.1: Die bipartiten Graphen des Tornado Codes

Wir nennen die „linken“ Knoten von  $B_i$  (also die Knoten in  $U_i$ ) *Informationsbits* und die „rechten“ Knoten von  $B_i$  (also die Knoten in  $V_i$ ) *Check-Bits*. Die Informationsbits von  $B_1$  entsprechen den Eingabebits, wobei die Zuordnung von Eingabebits und Informationsbits zufällig gewählt ist. (Natürlich bleiben die Pakete mit einem Zeitstempel versehen).

Für die Kodierung der Eingabebits werden zuerst die Werte aller Checkbits von  $B_1$  durch *Xoring* der inzidenten Informationsbits von  $B_1$  berechnet. Sodann interpretieren wir die rechten Bits von  $B_1$  als die Informationsbits von  $B_2$  und berechnen die Checkbits von  $B_2$ , also die rechten Knoten von  $B_2$ , wiederum durch *Xoring* der inzidenten Informationsbits von  $B_2$ . Dieses Verfahren wird für alle bipartiten Graphen wiederholt.

Nach hinreichender Kompression (d.h. wenn  $|V_k| \leq \sqrt{n}$  zum ersten Mal erreicht wurde) wird der Reed-Solomon Code<sup>1</sup>  $D$  als letzte Kodierungsstufe der Hierarchie eingefügt. Die Folge der Informations- und Checkbits, zusammen mit den Namen der entsprechenden Knoten, ist die Kodierung der Eingabe.

#### Aufgabe 57

Ein linearer  $(m, n)$ -Code mit  $m \geq n$  wird durch eine  $0, 1$  Matrix  $A$  mit  $m$  Zeilen und  $n$  Spalten sowie  $\text{Rang}(A) = n$  (über  $\mathbb{Z}_2$ ) beschrieben. Eine Nachricht  $x \in \{0, 1\}^n$  wird dann durch  $y = Ax$  über dem Körper  $\mathbb{Z}_2$  kodiert. (Beachte, dass der Tornado Code ein linearer Code ist.)

<sup>1</sup>Ein fehlerkorrigierender Code, der in quadratischer Zeit kodiert und dekodiert.

Eine kodierte Nachricht  $y$  wird durch den Erasure Kanal geschickt und man erhält die Nachricht  $y'$  mit

$$y'_i = \begin{cases} y_i & \text{Bit } i \text{ „überlebt“,} \\ * & \text{sonst.} \end{cases}$$

Zeige: Wenn  $y'$  nur von einer Nachricht  $x$  „erzeugt“ wurde, dann kann  $x$  in  $\text{poly}(n)$  Schritten bestimmt werden.

**Fazit:** Die Dekodierung linearer Erasure Codes ist also effizient möglich. Da sogar zufällig ausgewürfelte Matrizen  $A$  ihren maximalen Rank nach (zufälligem) Streichen von (nicht zu vielen) Zeilen (hochwahrscheinlich) beibehalten, sind lineare Codes gute Erasure-Codes. Tornado Codes sind aber durch ihre superschnelle Kodierung und Dekodierung unter den linearen Codes ausgezeichnet.

**Algorithmus 6.1** Erzeugung des Tornado Codes und Kodierung.

**Komponenten:** die Eingabelänge  $n$ , der Check-Faktor  $\beta$  und der Reed-Solomon Code  $D$ .

- (1) Wähle  $m$  maximal so, dass  $n \cdot \beta^m \geq \sqrt{n}$ . Erzeuge die bipartiten Graphen  $B_i = (U_i, V_i, E_i)$  (für  $i = 1, \dots, m+1$ ).

- (a) Bestimme Grad-Sequenzen  $u = (u_{i,2}, \dots, u_{i,L})$  und  $v = (v_{i,2}, \dots, v_{i,R})$  (mit der Intention, dass  $U_i$ , bzw.  $V_i$ , genau  $u_{i,j}$ , bzw.  $v_{i,j}$ , Knoten vom Grad  $j$  besitzt).  $B_i$  wird dann  $e_i = \sum_{j=2}^L j \cdot u_{i,j} = \sum_{j=2}^R j \cdot v_{i,j}$  Kanten erhalten.

/\* Grad-Sequenzen werden später berechnet. \*/

- (b) Setze  $U_i^* = \{1, \dots, e_i\} = V_i^*$  und bestimme eine zufällige Bijektion  $b_i : U_i^* \rightarrow V_i^*$ .

- (c) Zerlege  $U_i^*$  in beliebiger Weise in  $u_{i,j}$  disjunkte Teilmengen der Größe  $j$  (für  $j = 2, \dots, L$ ) und zerlege  $V_i^*$  in beliebiger Weise in  $v_{i,j}$  disjunkte Teilmengen der Größe  $j$  (für  $j = 2, \dots, R$ ). „Kollabiere“ die erhaltenen Teilmengen auf jeweils einen Repräsentantenknoten und nenne die neuen Knotenmengen  $U_i$  und  $V_i$ .

/\* Damit haben wir alle Knoten in  $U_i$ , bzw. in  $V_i$  vom Grad  $j$  definiert. \*/

- (d) Verbinde einen Knoten  $u \in U_i$  mit einem Knoten in  $v \in V_i$ , wenn ein Element der Teilmenge von  $u$  auf ein Element der Teilmenge von  $v$  durch die Bijektion  $b$  abgebildet wird.

/\* Bei diesem Prozess können Mehrfach-Kanten auftreten, die auch als solche zu behandeln sind. Beachte, dass  $U_i$  (bzw.  $V_i$ ) genau  $u_{i,j}$  (bzw.  $v_{i,j}$ ) Knoten vom Grad  $j$  besitzt. \*/

- (2) Permutiere die Eingabefolge zufällig. (Natürlich bleiben die Pakete mit einem Zeitstempel versehen). Kodiere die permutierte Folge, indem zuerst die Eingabebits an die linken Knoten von  $B_1$  angelegt werden. Sodann berechne die Werte aller Check-Bits. Nach der Kodierung gemäß  $B_{m+1}$  ist die Kodierung gemäß  $D$  die letzte Phase der Kodierung. (Die Codewortlänge von  $D$  sei  $\Theta(\beta^{m+1} \cdot n)$ .)

Die Folge der Informations- und Checkbits, zusammen mit den Namen der entsprechenden Knoten, ist die Kodierung der Eingabe.

### Aufgabe 58

Die Tornado-Codes kodieren die Menge der  $n$  Eingabebits Bits durch Einfügen von Checkbits mit dem Check-Faktor  $\beta$ . Der Code endet mit  $\sqrt{n}$  Checkbits, die durch den Reed-Solomon Code in quadratischer Zeit (in  $\sqrt{n}$ ) kodiert und dekodiert werden.

Wie groß sollte  $\beta$  (in Abhängigkeit von der Löschwahrscheinlichkeit  $\rho$ ) mindestens sein, um eine Rekonstruktion nicht auszuschliessen?

Wir werden später einen kleinen linearen Anteil weiterer Knoten und Kanten einsetzen. Wir haben in der Beschreibung des Tornado Codes die Konstruktion der Grad-Sequenzen offengelassen. Die Grad-Sequenzen werden wir so bestimmen, dass der folgende Dekodierungsalgorithmus eine möglichst große Fehlerrate verkräftet.

**Algorithmus 6.2** Dekodier-Algorithmus für Tornado Codes.

- (1) Die kodierte Folge ist nach Verlust einiger Bits zu dekodieren.
- (2) Führe die Fehlerkorrektur gemäß Code  $D$  durch. Setze  $i = m + 1$ .
- (3) Wiederhole, solange es mindestens ein Check-Bit  $c$  in  $B_i$  mit **genau einem** fehlenden Vorgänger  $b_0$  gibt:

Setze  $b_0 = c \oplus b_1 \oplus \dots \oplus b_k$ , wobei  $b_1, \dots, b_k$  die vorhandenen Vorgänger von  $c$  seien.

- (4) Wenn ein linker Knoten von  $B_i$  nicht korrigiert werden konnte, dann brich mit einer Fehlermeldung ab.

Ansonsten, wenn  $i = 1$ , dann ist die Korrektur geglückt. Wenn  $i > 1$ , dann setze  $i = i - 1$  und wiederhole Schritt (3).

**Aufgabe 59**

Die bipartiten Graphen in der Definition des Tornado Codes mögen die Knotenmenge  $V$  und die Kantenmenge  $E$  besitzen.

Beschreibe, wie Algorithmus 6.2 in Zeit  $O(|V| + |E|)$  implementiert werden kann. Benenne insbesondere geeignete Datenstrukturen.

**Aufgabe 60**

Algorithmus 6.2 gibt in einigen Fällen auf, obwohl eine Rekonstruktion möglich wäre.

- (a) Gib einen bipartiten Graphen  $G = (L \cup R, E)$  an und weise jedem Knoten einen Wert aus  $\{0, 1, *\}$  zu, so dass
  - jeder Knoten aus  $R$  einen Wert aus  $\{0, 1\}$  erhält,
  - es genau eine Belegung der mit  $*$  beschrifteten Knoten mit  $\{0, 1\}$  gibt, so dass die Beschriftung aller Knoten aus  $R$  der Parität der Beschriftungen seiner Nachbarn entspricht,
  - aber Algorithmus 6.2 die eindeutige Lösung nicht findet.
- (b) Gib zwei bipartite Graphen  $G_1 = (L_1 \cup R_1, E_1)$  und  $G_2 = (L_2 \cup R_2, E_2)$  mit  $R_1 = R_2$  nebst  $\{0, 1, *\}$  Beschriftung an, die die folgenden Eigenschaften erfüllen.
  - Jeder Knoten aus  $R_2$  hat einen Wert aus  $\{0, 1\}$ .
  - Keiner der Graphen  $G_i$  besitzt eine eindeutige Belegung der mit  $*$  beschrifteten Knoten mit Werten aus  $\{0, 1\}$ , so dass die Beschriftung jedes Knotens aus  $R_i$  der Parität der Beschriftung seiner Nachbarn in  $L_i$  entspricht.
  - Der verklebte Graph  $G = (L_1 \cup L_2 \cup R_2, E_1 \cup E_2)$  besitzt eine eindeutige Belegung der mit  $*$  beschrifteten Knoten mit Werten aus  $\{0, 1\}$ , so dass die Beschriftung jedes Knotens aus  $R_i$  der Parität der Beschriftung seiner Nachbarn in  $L_i$  entspricht.
- (c) Warum ersetzt man die Heuristik von Algorithmus 6.2 nicht durch die Gauß'sche Eliminierung?

**Aufgabe 61**

$G = (L \cup R, E)$  sei ein bipartiter Graph mit maximalem Grad zwei für die Knoten in  $R$ . Die Werte aller Knoten von  $R$  seien bekannt, während die Werte einiger linker Knoten fehlen.

Beweise oder widerlege: Algorithmus 6.2 findet **stets** eine eindeutige konsistente Beschriftung, wenn eine solche existiert.

Das Abbruchkriterium in Schritt (4) für  $i > 1$  dient nur der theoretischen Analyse. In praktischen Anwendungen sollte ein Abbruch nur erfolgen, wenn tatsächlich Informationsbits der *ersten* Hierarchie nicht rekonstruiert wurden.

Warum wählen wir nicht reguläre bipartite Graphen, also Graphen in denen alle linken Knoten denselben Grad und alle rechten Knoten denselben Grad besitzen? Vom Standpunkt eines

Informationsbits (also vom Standpunkt eines linken Knotens) ist ein hoher Grad hochwillkommen, denn dann bestehen die größten Chancen von irgendeinem der vielen rechten Nachbarn bestimmt zu werden. Andererseits werden diese Chancen zunichte gemacht, wenn alle linken Nachbarn denselben hohen Grad besitzen: Nur wenige rechte Knoten besitzen dann genau einen fehlenden linken Nachbarn.

Bei einer sorgfältigen Auswahl der Gradsequenzen (mit dementsprechender nicht-regulärer Struktur) wird die Rekonstruktion in Wellen verlaufen. Die wenigen hochgradigen linken Nachbarn werden sofort rekonstruiert, gefolgt von ihren etwas niedrig-gradigeren Kollegen und so weiter. Die vielen niedrigst-gradigen linken Knoten werden dann in wenigen letzten Wellen rekonstruiert: Der Tornado produziert rasant wachsende Wellen rekonstruierter linker Knoten. Die in Kapitel 6.2 durchgeführte Analyse wird diese Analogie begründen.

## 6.1 Informationsrate

Wie kompakt ist die Kodierung? Wir definieren die Informationsrate durch

$$\text{Informationsrate} = \frac{\text{Eingabelänge}}{\text{Codewortlänge}}.$$

Die Codewortlänge beträgt

$$n + \sum_{i=1}^m \beta^i \cdot n + O(\sqrt{n}) = n \cdot \frac{1 - \beta^{m+1}}{1 - \beta} + O(\sqrt{n}) \leq \frac{n}{1 - \beta} + O(\sqrt{n}).$$

Da  $n$  Eingabebits vorliegen, erhalten wir die Informationsrate

$$\frac{1 - \beta}{1 + O\left(\frac{1 - \beta}{\sqrt{n}}\right)}.$$

Sei  $\rho$  die Wahrscheinlichkeit, mit der der Kanal Bits verliert. Dann ist offensichtlich eine Codewortlänge von mindestens  $\frac{n}{1 - \rho}$  Bits notwendig, um den Bitverlust des Kanals zu kompensieren, und damit wird die Informationsrate bestenfalls  $1 - \rho$  betragen.

Wir machen den Ansatz  $\beta = \rho(1 + \varepsilon)$  und erreichen damit erstaunlicherweise eine Informationsrate von fast  $1 - \rho$ . Es bleibt zu zeigen, dass eine Rekonstruktion in jeder Hierarchiestufe  $B_i$  mit hoher Wahrscheinlichkeit gelingt. Dabei können wir rekursiv annehmen, dass alle Check-Bits von  $B_i$  bereits rekonstruiert wurden. (Die Induktionsbasis wird durch den fehlerkorrigierenden Code  $D$  gelegt.)

## 6.2 Berechnung der Grad-Sequenzen

Wir stellen uns vor, dass ein durch das Löschen von Bits verstümmeltes Codewort  $w$  zu dekodieren ist. Die Wahrscheinlichkeit des Löschens einer Bitposition sei  $\rho$  und die Löschungen der Positionen seien stochastisch unabhängig. Wir nehmen weiterhin an, dass die Rückwärtsdekodierung von Algorithmus 6.2 den Graphen  $B = B_i$  erreicht hat. Das setzt voraus, dass ein jeder rechter Knoten von  $B_i$  entweder bereits vorhanden ist oder erfolgreich rekonstruiert wurde. Wir sagen, dass ein rechter Knoten von  $B$  *rekonstruiert*, falls höchstens einer seiner linken Nachbarn verloren gegangen ist. Sei  $v$  ein beliebiger linker Knoten von  $B$ , dessen Bit verloren gegangen ist. Wann werden wir im Stande sein, das verlorene Bit zu bestimmen?

Zuerst betrachten wir sämtliche Check-Knoten, die mit  $v$  benachbart sind und hoffen auf einen benachbarten rekonstruierenden Check-Knoten. Formal können wir somit  $v$  als einen *ODER-Knoten* interpretieren, der auf *mindestens einen* rekonstruierenden Nachbarn hofft. Wann kann uns einer dieser (rechten) Nachbarn  $w$  glücklich machen? Genau dann, wenn *alle* linken Nachbarn von  $w$  (mit der Ausnahme von  $v$ ) vorhanden sind. Mit anderen Worten, jeder rechte Nachbar  $w$  von  $v$  spielt die Rolle eines *UND-Knotens*. Wir haben aber noch nicht verloren, wenn sich kein rekonstruierender rechter Nachbar von  $v$  findet, denn „wer noch kein rekonstruierender Knoten ist, der kann’s noch werden“! Nach der ersten Generation (der rechten Nachbarn von  $v$ ), schauen wir uns deshalb die zweite Generation, also die Enkelkinder von  $v$ , an. Nicht vorhandene Enkelkinder von  $v$  können mit der gleichen Überlegung wie für  $v$  rekonstruiert werden: auch sie lassen sich formal als ODER-Knoten auffassen.

Was passiert, wenn wir diesen Prozess solange wiederholen, bis wir alle Knoten im Abstand  $2l$  von  $v$  erreicht haben? (Wir werden  $l$  so klein wählen, dass der von den erreichten Knoten induzierte Teilgraph von  $B$  mit hoher Wahrscheinlichkeit ein Baum ist.) Wir setzen  $v$  als ODER-Knoten auf die Wurzel des Baums, gefolgt von den rechten Nachbarn (als UND-Knoten), gefolgt von deren linken Nachbarn (mit Ausnahme von  $v$ ) als ODER-Knoten und so weiter. Die Blätter dieses Baums  $T_l(v)$  sind linke Knoten, die wir mit 0 (nicht vorhanden) oder 1 (vorhanden) beschriften. Die Knoten der Tiefe  $2l - 1$  sind rechte Knoten, die genau dann rekonstruierend werden, wenn alle Kinder vorhanden sind, d.h. mit 1 beschriftet sind. Mit anderen Worten, ein Knoten der Tiefe  $2l - 1$  ist rekonstruierend, wenn der Knoten als UND-Knoten den Wert 1 berechnet. Wann ist ein Knoten der Tiefe  $2l - 2$  rekonstruierbar, bzw. vorhanden? Um dies zu formalisieren, erlauben wir das Festfrieren des entsprechenden ODER-Knotens auf 1 zusätzlich zum Formalismus des UND-ODER Baums (falls das entsprechende Paket nicht verloren ist). Damit ist ein Knoten der Tiefe  $2l - 2$  genau dann rekonstruierbar oder vorhanden, wenn der Knoten festgefroren ist, oder wenn eines seiner Kinder rekonstruierend ist. Und die Berechnung des Wertes 1 ist hinreichend und notwendig zur Berechnung des Bits des Knotens.

Wir fassen zusammen. Mit einem induktiven Argument erhalten wir, dass der Ausgangsknoten  $v$  genau dann entweder vorhanden ist (also festgefroren auf Wert 1) oder rekonstruierbar ist, wenn der oben beschriebene UND-ODER Baum  $T_l(v)$  den Wert 1 berechnet!

Schauen wir uns die möglichen Bäume  $T_l(v)$  genauer an. Wir wollen die Wahrscheinlichkeit in  $B$  bestimmen, dass eine Kante einen linken (bzw. rechten) Knoten mit einem rechten (bzw. linken) Knoten vom Grad  $j$  verbindet. Wir besitzen insgesamt  $\sum_{j=2}^L j \cdot u_{i,j} = \sum_{j=2}^R j \cdot v_{i,j}$  Kanten. Von diesen Kanten verbinden genau  $j \cdot v_{i,j}$  Kanten einen linken Knoten mit einem rechten Knoten vom Grad  $j$ , und genau  $j \cdot u_{i,j}$  Kanten verbinden einen rechten Knoten mit einem linken Knoten vom Grad  $j$ . Wir definieren die Wahrscheinlichkeiten

$$p_j = \frac{j \cdot u_{i,j}}{\sum_{j=2}^L j \cdot u_{i,j}} \quad \text{und} \quad q_j = \frac{j \cdot v_{i,j}}{\sum_{j=2}^R j \cdot v_{i,j}}$$

dafür, dass eine Kante einen rechten (bzw. linken) Knoten mit einem linken (bzw. rechten) Knoten vom Grad  $j$  verbindet. Diese Verteilungen definieren auch den durchschnittlichen linken (bzw. rechten) Grad. Zum Beispiel ist

$$\sum_{j=2}^L \frac{p_j}{j} = \frac{\sum_{j=2}^L u_{i,j}}{\sum_{j=2}^L j \cdot u_{i,j}} = \frac{1}{\text{Grad}_{\text{links}}}. \quad (6.1)$$

Beachte, dass stets

$$\beta \cdot \sum_{j=2}^L \frac{p_j}{j} = \frac{\beta \cdot n}{\text{Kantenzahl}} = \sum_{j=2}^R \frac{q_j}{j} \quad (6.2)$$

gilt.

Wenn wir uns einen beliebigen Knoten  $v$  in  $T_l(v)$  ungerader Tiefe (also einen rechten Knoten) anschauen und eine Kante  $e = (v, u)$  zu einem Kind  $u$  wählen, dann verbindet  $e$  den Knoten  $v$  mit Wahrscheinlichkeit  $p_j$  mit einem Knoten vom Grad  $j$ : Also wird ein innerer Knoten  $u$  in gerader Tiefe genau  $j - 1$  Kinder mit Wahrscheinlichkeit  $p_j$  besitzen (beachte, dass eine Kante von  $u$  bereits vom Vater geschluckt wurde).

Ist  $u$  hingegen ein beliebiger Knoten ungerader Tiefe in  $T_l(v)$ , dann wird  $u$  mit Wahrscheinlichkeit  $q_j$  genau  $j - 1$  Kinder besitzen. Leider bringt die Wurzel diesen Formalismus durcheinander, da sie keinen Vater besitzt; deshalb stiebitzen wir eine ihrer Kanten durch zufällige Wahl.

Hier ist allerdings ein Wort der **Warnung** angebracht. Die Kinderzahlen entsprechen nicht unabhängigen Zufallsvariablen: Wenn bereits viele Knoten mit einer bestimmten Kinderzahl im Baum aufgetreten sind, wird diese Kinderzahl entsprechend unwahrscheinlicher, denn unsere bipartiten Graphen besitzen eine festgelegte Anzahl von Knoten für jeden Grad. Insbesondere sind die zugewiesenen Wahrscheinlichkeiten für die Kinderzahlen nur als approximativ anzusehen. Wir werden deshalb dafür Sorge tragen, dass die Bäume  $T_l(v)$  klein bleiben, damit die Approximation nur mit einem kleinen, vernachlässigbaren Fehler behaftet ist.

Um die möglichen UND-ODER Bäume  $T_l(v)$  zu beschreiben, führen wir die Klasse  $\mathcal{T}_l(\vec{p}, \vec{q})$  von UND-ODER Bäumen der Tiefe  $2l$  ein, um die zufällige Wahl der bipartiten Graphen wiederzugeben. Wenn wir uns vorstellen, dass wir die Bäume in  $\mathcal{T}_l(\vec{p}, \vec{q})$  von der Wurzel aus wachsen lassen, dann wird ein ODER-Knoten  $v_{\text{ODER}}$  mit Wahrscheinlichkeit  $p_i$  genau  $i - 1$  Kinder und ein UND-Knoten  $v_{\text{UND}}$  mit Wahrscheinlichkeit  $q_i$  genau  $i - 1$  Kinder besitzen. (Wir modellieren also die zufällige Konstruktion der bipartiten Graphen.)

Wir nehmen jetzt auch die Löschwahrscheinlichkeit  $\rho$  mit in unseren Formalismus auf. Dazu markiere die Blätter eines Baums in  $\mathcal{T}_l(\vec{p}, \vec{q})$  zufällig durch 0 (mit Wahrscheinlichkeit  $\rho$ ) oder durch 1 (mit Wahrscheinlichkeit  $1 - \rho$ ). Linke Knoten, also Knoten gerader Tiefe, werden ebenfalls mit Wahrscheinlichkeit  $1 - \rho$  auf den Wert 1 festgefroren. (Wir modellieren den zufällig löschenden Erasure Kanal.)

Im **Evaluierungsproblem für UND-ODER Bäume** fragt man nach der Wahrscheinlichkeit  $\text{Null}_l(\rho)$ , dass die angelegte zufällige Eingabe die Ausgabe 0 an der Wurzel produziert. (Der Wahrscheinlichkeitsraum besteht aus allen Paaren  $(T, x)$ , so dass  $x$  eine Eingabe für den Baum  $T \in \mathcal{T}_l(\vec{p}, \vec{q})$  ist. Die Wahrscheinlichkeit des Paares  $(T, x)$  ist das Produkt der Wahrscheinlichkeit von  $T$  mit der Wahrscheinlichkeit von  $x$ . Die Wahrscheinlichkeit für eine Null an einer beliebigen Position in  $x$  ist  $\rho$ .)

Erstaunlicherweise lässt sich die Wahrscheinlichkeit  $\text{Null}_l(\rho)$  leicht rekursiv bestimmen. Dazu zuerst eine vorbereitende Definition.

**Definition 6.3** Für die Verteilungen  $\vec{p} = (p_1, \dots, p_L)$  und  $\vec{q} = (q_1, \dots, q_R)$  definieren wir die Polynome

$$p(x) = \sum_{i=2}^L p_i \cdot x^{i-1} \quad \text{und} \quad q(x) = \sum_{i=2}^R q_i \cdot x^{i-1}$$

sowie das Iterationspolynom

$$f(x) = \rho \cdot p(1 - q(1 - x)).$$

**Lemma 6.4** Setze  $\text{Null}_0(\rho) = \rho$ . Dann gilt für jedes  $l \geq 1$ ,

$$\text{Null}_{l+1}(\rho) = f(\text{Null}_l(\rho)).$$

**Beweis:** Es ist

$$f(\text{Null}_l(\rho)) = \rho \cdot p(1 - q(1 - \text{Null}_l(\rho))).$$

Wir betrachten ein UND-Gatter der Tiefe 1. Wenn das UND-Gatter  $i - 1$  Kinder besitzt (und dies passiert mit Wahrscheinlichkeit  $q_i$ ), dann kann nur dann der Wert 1 erzielt werden, wenn alle  $i - 1$  ODER-Kinder den Wert 1 erreichen und dies passiert mit Wahrscheinlichkeit  $(1 - \text{Null}_l(\rho))^{i-1}$ . Mit anderen Worten,  $q(1 - \text{Null}_l(\rho))$  ist die Wahrscheinlichkeit, dass ein UND-Knoten der Tiefe 1 wahr wird.

Wann wird die Wurzel den Wert 0 bekommen? Zuerst darf die Wurzel nicht festgefroren sein (und dies passiert mit Wahrscheinlichkeit  $\rho$ ). Wenn das ODER-Gatter  $j - 1$  Kinder besitzt, darf weiterhin keines dieser Kinder wahr sein und wir wissen bereits, dass dies mit Wahrscheinlichkeit  $(1 - q(1 - \text{Null}_l(\rho)))^{j-1}$  passiert. Die Wahrscheinlichkeit für  $j - 1$  Kinder ist aber  $p_j$  und damit ist  $\rho \cdot p(1 - q(1 - \text{Null}_l(\rho)))$  die Wahrscheinlichkeit für das Wurzelergebnis 0 und die Behauptung folgt.  $\square$

Wir möchten erreichen, dass  $\text{Null}_l(\rho) = f^{(l)}(\rho)$  schnell kleine Werte annimmt. Dazu machen wir den Ansatz

$$f(x) = \rho \cdot p(1 - q(1 - x)) < x \quad \text{für } 0 < x \leq \rho. \quad (6.3)$$

Wenn Bedingung (6.3) gilt, dann wird die Folge  $f^{(l)}(\rho)$  in  $l$  monoton fallen und gegen Null konvergieren. Wenn andererseits die Bedingung für  $x_0 \leq \rho$  verletzt ist, gilt  $f^{(l)}(x_0) > x_0$  für jedes  $l$ . (Dies folgt, da die Polynome  $p(x)$  und  $q(x)$  monoton wachsende Funktionen in  $x$  sind. Damit ist auch  $f$  eine monoton wachsende Funktion in  $x$ .)

Um die Gradverteilung der linken Knoten zu definieren, legen wir die folgende Verteilung fest:

$$p_i = \frac{1}{(i-1) \cdot \sum_{i=1}^d \frac{1}{i}} \quad \text{für } i = 2, \dots, d+1.$$

Um den durchschnittlichen Grad der linken Knoten zu berechnen, beachte

$$\sum_{j=2}^{d+1} \frac{p_j}{j} = \frac{1}{\sum_{i=1}^d \frac{1}{i}} \cdot \sum_{j=2}^{d+1} \frac{1}{(j-1) \cdot j} = \frac{1}{\sum_{i=1}^d \frac{1}{i}} \cdot \frac{d}{d+1}$$

und mit (6.1) folgt  $\text{Grad}_{\text{links}} = \sum_{i=1}^d \frac{1}{i} \cdot \frac{d+1}{d}$ . Wir bestimmen ebenso die Gradverteilung der rechten Knoten durch die Verteilung

$$q_i = \frac{e^{-\alpha} \cdot \alpha^{i-1}}{(i-1)!} \quad \text{für } i \geq 2.$$

Um die Beziehung (6.2) einzuhalten, müssen wir  $\alpha$  so wählen, dass

$$\sum_{i \geq 2} \frac{q_i}{i} = \sum_{i \geq 2} \frac{e^{-\alpha} \cdot \alpha^{i-1}}{i!} = \frac{e^{-\alpha}}{\alpha} \cdot (e^\alpha - 1) = \frac{\beta}{\text{Grad}_{\text{links}}}$$

gilt. Diese Forderung wird erfüllt, falls

$$\frac{\text{Grad}_{\text{links}}}{\beta} = \frac{\alpha \cdot e^\alpha}{e^\alpha - 1}$$

gilt. Wir haben  $\text{Grad}_{\text{links}}$  bereits berechnet und erhalten deshalb

$$\alpha = \frac{\text{Grad}_{\text{links}}}{\beta} \cdot \frac{e^\alpha - 1}{e^\alpha} = \sum_{i=1}^d \frac{1}{i} \cdot \frac{d+1}{d} \cdot \frac{e^\alpha - 1}{\beta \cdot e^\alpha}$$

Wir haben hier ein wenig geschummelt, da die tatsächliche Verteilung auf den rechten Knoten natürlich auch endlich sein muss. Da die  $q_i$  aber schnell gegen Null konvergieren, haben wir nur einen vernachlässigbaren Fehler eingebaut.

Wir können jetzt überprüfen, wie schnell  $\text{Null}_l(\rho)$  gegen Null konvergiert. Es ist

$$p(x) < \frac{-1}{\sum_{i=1}^d \frac{1}{i}} \cdot \ln(1-x) \quad \text{und} \quad q(x) = e^{\alpha \cdot x - \alpha}$$

und deshalb

$$f(x) = \rho \cdot p(1 - q(1-x)) < \frac{-\rho}{\sum_{i=1}^d \frac{1}{i}} \cdot \ln(e^{\alpha \cdot (1-x) - \alpha}) = \frac{\rho}{\sum_{i=1}^d \frac{1}{i}} \cdot \alpha \cdot x.$$

Mit der Forderung an  $\alpha$  erhalten wir

$$f(x) < \frac{\rho \cdot (d+1)}{\beta \cdot d} \cdot \frac{e^\alpha - 1}{e^\alpha} \cdot x$$

und für  $\rho \leq \beta \cdot \frac{d}{d+1}$  ist insbesondere  $f(x) < \frac{e^\alpha - 1}{e^\alpha} \cdot x$ . Als Konsequenz:

**Lemma 6.5** Für die Verteilungen  $\vec{p}$  und  $\vec{q}$  mit  $p_i = \frac{1}{(i-1) \cdot \sum_{i=1}^d \frac{1}{i}}$  (für  $i = 2, \dots, d+1$ ) und  $q_i = \frac{e^{-\alpha} \cdot \alpha^{i-1}}{(i-1)!}$  (für  $i \geq 2$ ) ist Bedingung (6.3) erfüllt, falls

$$\rho \leq \beta \cdot \frac{d}{d+1}.$$

### 6.3 Leistungsdaten der Tornado Codes

Wir nehmen von jetzt ab an, dass die bisher nicht fixierten Gradsequenzen gemäß den Verteilungen  $\vec{p}$  und  $\vec{q}$  festgelegt werden. Nach Lemma 6.4 sowie Lemma 6.5 ist damit die Wahrscheinlichkeit der erfolglosen Rekonstruktion eines Bits an einer **vorher fixierten** Bitposition durch eine beliebig kleine Konstante nach oben beschränkt.

Aber wir besitzen  $n$  viele Bitpositionen und die erfolglose Rekonstruktion eines wenn auch kleinen, aber immer noch linear großen Anteils der Bitpositionen kann damit nicht ausgeschlossen werden! In experimentellen Tests wurden dann auch häufig nicht-rekonstruierbare Bitpositionen gefunden.

Aber wir haben dennoch einen signifikanten Fortschritt gemacht, denn der erwartete Prozentsatz  $\rho$  von fehlenden Bits konnte auf einen wesentlich kleineren linearen Anteil  $\alpha$  nicht-rekonstruierbarer Bits gesenkt werden. Darüberhinaus kann gezeigt werden, dass  $B_i$  mit

Wahrscheinlichkeit mindestens  $1 - 2^{-\Omega(\beta^i \cdot n)}$  höchstens  $\alpha \cdot \beta^i \cdot n$  nicht rekonstruierbare Informationsbits besitzt.

Der Rekonstruktions-Prozess beginnt mit dem fehlerkorrigierenden Code  $D$  mit Codewortlänge  $\Theta(\sqrt{n})$ .  $D$  sei so gewählt, dass  $D$  einen Prozentsatz von  $\rho' > \rho$  fehlenden Bits mit konstanter Fehlerrate verkräftet. Damit wird  $D$  mit Wahrscheinlichkeit  $1 - 2^{-\Omega(\sqrt{n})}$  eine vollständige Rekonstruktion erlauben, denn mit dieser Wahrscheinlichkeit wird ein Prozentsatz von höchstens  $\rho'$  aller Bits fehlen. Damit sind sämtliche Check-Bits des letzten bipartiten Codes  $B_m$  rekonstruiert und leider wird dennoch ein  $\alpha$ -Anteil der Informationsbits von  $B_m$  nicht rekonstruierbar sein.

Wir fügen einen relativ kleinen, aber immer noch linear großen Anteil von **neuen** Check-Bits zu  $B_m$  hinzu und verbinden die  $\beta^m \cdot n$  Informationsbits von  $B_m$  mit diesen neuen Check-Bits über linear viele, zufällig eingesetzte Kanten. Der neue bipartite Graph wird auf allen Informationsbits von  $B_m$  den Grad  $g$  besitzen.

Wir geben eine informelle Beschreibung des Effekts dieser neuen Kanten. Sei  $U'$  die Menge nicht-rekonstruierbarer Informationsbits, also  $|U'| \leq \alpha \cdot \beta^m \cdot n$ . Da wir Kanten zufällig eingesetzt haben, wird  $U'$  mindestens  $\frac{g}{2} \cdot |U'| + 1$  Nachbarn besitzen: Bei zufälliger Kantenauswahl wird die Menge  $U'$  also *expandieren*. Wenn jeder Nachbar mit mindestens zwei Knoten aus  $U'$  verbunden ist, dann kann  $U'$  höchstens  $\frac{g}{2} \cdot |U'|$  Nachbarn besitzen: Es gibt mindestens einen Nachbarn, der nur mit genau einem Knoten  $u \in U'$  verbunden ist! Dieser Knoten  $u$  ist jetzt rekonstruierbar! Die Wiederholung dieses Arguments garantiert die vollständige Rekonstruktion aller Informationsbits von  $B_m$ .

Damit sind aber alle Check-Bits von  $B_{m-1}$  rekonstruiert und wir wiederholen unser Vorgehen diesmal für  $B_{m-1}$ .

### Aufgabe 62

Ein bipartiter Graph  $G = (L \cup R, E)$  heißt ein  $(\gamma, \alpha)$ -Expander, wenn für *alle*  $U \subseteq L$  mit  $|U| \leq \alpha \cdot |L|$  gilt  $|\Gamma(U)| \geq \gamma \cdot |U|$ , wobei  $\Gamma(U)$  die Menge der mit  $U$  benachbarten Knoten ist.

Ein bipartiter Graph  $G = (L \cup R, E)$  heißt ein *eindeutiger*  $\alpha$ -Expander, wenn für alle  $U \subseteq L$  mit  $|U| \leq \alpha \cdot |L|$  ein Knoten  $v \in R$  mit genau einem Nachbarn in  $U$  existiert.

- a)  $G = (L \cup R, E)$  sei ein durch folgenden Prozess zufällig erzeugter bipartiter Graph mit  $|L| = |R| = n$ : Zu jedem Knoten  $u \in L$  wird zufällig gleichverteilt mit Wahrscheinlichkeit  $\binom{n}{d}^{-1}$  die Nachbarmenge  $\Gamma(\{u\}) \subseteq R$  mit  $|\Gamma(\{u\})| = d$  gewählt.

$\alpha$  sei eine hinreichend kleine Zahl. Zeige, dass  $G$  mit positiver Wahrscheinlichkeit ein  $(\frac{d}{2}, \alpha)$ -Expander ist. Diese Aussage ist für  $d = O(1)$  zu zeigen.

**Hinweis:** Zeige, dass die Wahrscheinlichkeit, dass  $G$  eine Menge  $U$  mit  $\emptyset \neq U \subseteq L$  und  $|U| \leq \alpha n$  und eine Menge  $W$  mit  $\Gamma(U) \subseteq W \subseteq R$  und  $|W| = \frac{d}{2}|U|$  enthält, klein ist.

Die Stirling Approximation liefert  $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$ .

- b) Zeige, dass ein  $(\gamma, \alpha)$ -Expander auch ein eindeutiger  $\alpha$ -Expander ist, falls  $\gamma > \frac{d}{2}$ , wobei jeder Knoten  $v \in L$  den Grad  $d$  besitze.

**Satz 6.6** Die Verteilungen  $\vec{p}$  und  $\vec{q}$  seien durch  $p_i = \frac{1}{(i-1) \cdot \sum_{i=1}^d \frac{1}{i}}$  (für  $i = 2, \dots, d+1$ ) und  $q_i = \frac{e^{-\alpha} \cdot \alpha^{i-1}}{(i-1)!}$  (für  $i \geq 2$ ) definiert.  $D$  sei ein „guter“ fehlerkorrigierender Code.  $D$  habe die Codewortlänge  $\Theta(\sqrt{n})$  und besitze Kodierungs- und Dekodierungsalgorithmen mit höchstens quadratischer Laufzeit.

Der Tornado Code  $T$  werde durch  $D$ , die Verteilungen  $\vec{p}$  und  $\vec{q}$ , den Parameter  $\beta = \rho \cdot (1 + \frac{1}{2 \cdot \varepsilon})$  und durch die Einsetzung von Expander Graphen entsprechender Größe konstruiert. Dann besitzt  $T$  für  $\rho < \frac{1}{2}$  die Informationsrate

$$\text{Rate}(T) \geq (1 - \rho) \cdot (1 - \varepsilon).$$

Eine Rekonstruktion gelingt für hinreichend großes  $n$  mit Wahrscheinlichkeit mindestens  $1 - 2^{-\Omega(\sqrt{n})}$ . Kodierungs- und Dekodierungsalgorithmen benötigen höchstens  $O(n \cdot \ln(\frac{1}{\varepsilon}))$  Schritte.

**Beweis:** Wir haben bereits die Informationsrate berechnet und

$$\frac{1 - \beta}{1 + O(\frac{1 - \beta}{\sqrt{n}})}$$

als Ergebnis (vor dem Einsetzen der Expandergraphen) erhalten. Diese Rate wird um den Faktor  $1 - \gamma$  für eine kleine Konstante  $\gamma$  sinken, da die Expander Graphen neue Knoten in den Graphen des Tornado Codes einfügen. Für hinreichend großes  $n$  kann  $\gamma$  als beliebig klein angenommen werden.

Für eine erfolgreiche Rekonstruktion genügt die Forderung  $\beta \geq \rho \cdot \frac{d+1}{d}$  gemäß Lemma 6.5. Wir setzen  $d = \frac{2}{\varepsilon}$  und erhalten die Forderung  $\beta \geq \rho \cdot (1 + \frac{\varepsilon}{2})$ , die wir natürlich mit  $\beta = \rho \cdot (1 + \frac{\varepsilon}{2})$  erfüllen. Die Informationsrate ist also mindestens

$$(1 - \gamma) \cdot \frac{1 - \rho \cdot (1 + \frac{\varepsilon}{2})}{1 + O(\frac{1 - \rho \cdot (1 + \frac{\varepsilon}{2})}{\sqrt{n}})} \geq (1 - \rho) \cdot (1 - \varepsilon),$$

für hinreichend großes  $n$ .

Die Laufzeit der Kodier- und Dekodieralgorithmen ist proportional zur Kantenzahl des Graphen des Tornado Codes. Wir besitzen  $O(n \cdot \frac{1}{1 - \beta})$  Knoten, wobei  $\text{Grad}_{\text{links}} = \sum_{i=1}^d \frac{1}{i} \cdot \frac{d+1}{d} = \Theta(\ln(d))$  der durchschnittliche Grad der linken Knoten ist. Die Kantenzahl ist somit

$$O(n \cdot \frac{1}{1 - \beta} \cdot \ln(\frac{1}{\varepsilon})) = O(n \cdot \ln(\frac{1}{\varepsilon})),$$

denn wegen  $\rho < 1/2$  wird  $\beta$  nicht zu nahe an 1 „gedrängt“. □

**Bemerkung 6.2** Unter den konventionellen Erasure-Codes ist der Reed-Solomon Code am erfolgreichsten. In experimentellen Vergleichen mit Varianten des Reed-Solomon Codes wurden Dateien der Größen 1 MB, 2MB, 4 MB, 8 MB und 16 MB bei einer Paketgröße von einem KB kodiert und dekodiert. Die Informationsrate 1/2 wurde zu Grunde gelegt. Bei den Laufzeiten (Kodieren/Dekodieren) ergab sich unter Auswahl der besten Reed-Solomon Variante

- 1 MB. Tornado: 0.26 Sek / 0.14 Sek; Reed-Solomon 93 Sek / 40.5 Sek.
- 2 MB. Tornado: 0.53 Sek / 0.19 Sek; Reed-Solomon 442 Sek / 199 Sek.

- 4 MB. Tornado: 1.06 Sek / 0.40 Sek; Reed-Solomon 1717 Sek / 800 Sek.
- 8 MB. Tornado: 2.13 Sek / 0.87 Sek; Reed-Solomon 6994 Sek / 3166 Sek.
- 16MB. Tornado: 4.33 Sek / 1.75 Sek; Reed-Solomon 30802 Sek / 13269 Sek.

Die Experimente wurden auf einer SUN Ultrasparc (167 MHz) mit 64 MB RAM von den Autoren der Tornado Codes durchgeführt. Diese Zahlen belegen eindeutig den Laufzeitvorteil der Tornado Codes.

Kommen wir zu den Nachteilen. Tornado Codes basieren auf einer rekursiven Paritäts-Kodierung und sind deshalb sehr anfällig für verfälschte Pakete. Diese Verfälschungen werden nicht nur nicht korrigiert, sondern sogar vervielfacht. Dies scheint ihren Einsatz für sicheres Internet Routing jedoch nicht zu behindern, da bei kleinen Paketgrößen der Paketverlust, jedoch nicht die Verfälschung von Paketen droht.

**Bemerkung 6.3** Raptor codes [Sh] stellen eine Verbesserung von Tornado Codes dar. Sie erlauben die Implementierung von "Digital Fountains": Ein Raptor Code kodiert eine Nachricht der Länge  $n$  in fast gleicher Länge. Die kodierte Nachricht wird durch eine Folge von "Kodiersymbolen" repräsentiert und ein Empfänger muss nur genügend viele Kodiersymbole in beliebiger Reihenfolge erhalten, um erfolgreich zu dekodieren. Diese Situation ist mit einem digitalen Sprngbrunnen vergleichbar: Um den Durst zu löschen, genügt ein Glas Wasser und es ist ohne Belang, welche Wassertropfen sich im Glas befinden.

## Kapitel 7

# End-to-End Congestion Control

Paketverluste entstehen aufgrund von Überlast, defekten Routern, Bitfehlern während der Übertragung oder nicht ausreichender Puffer-Kapazität des Empfängers. Weitere Fehlerquellen sind duplizierte Pakete sowie das Eintreffen von Paketen in falscher Reihenfolge.

### Beispiel 7.1 Active Queue Management (AQM)

AQM-Algorithmen entfernen sogar vorsätzlich Pakete als Teil einer Verkehrskontrolle. Warum sollte man Pakete entfernen, wenn dies nicht erzwungen ist? Der Sender entfernter Paket stellt fest, dass diese nicht ankommen und vermindert daraufhin (hoffentlich) freiwillig die Sendegeschwindigkeit.

Der AQM-Algorithmus **Drop-Tail** weist ankommende Pakete ab, falls die maximale Queuegröße überschritten wird. Paketstaus werden allerdings nur langsam aufgelöst, da Sender zu spät von überlaufenden Queues erfahren, und deshalb versucht der AQM-Algorithmus **RED** (Random Early Detection), die durchschnittliche Queue-Größe klein zu halten. (Eine große durchschnittliche Queue-Größe zwingt einen Router zwar nicht notwendigerweise in die Knie, erzwingt aber eine deutlich verlangsamte Paketzustellung.) RED arbeitet mit den Schwellenwerten  $MIN$  und  $MAX$  und bestimmt bei einer durchschnittlichen Queue-Größe  $M$  die monoton in  $M$  wachsende Entfernungswahrscheinlichkeit  $p_M$ . Für  $M \leq MIN$  werden keine Pakete entfernt ( $p_M = 0$ ), während für  $MIN \leq M \leq MAX$  ankommende Pakete mit Wahrscheinlichkeit  $p_M$  entfernt werden. Schliesslich werden für  $M \geq MAX$  auch in der Queue befindliche Pakete entfernt.

Da das Verwerfen der Pakete in Routern zufällig erfolgt, können durch RED unter Umständen eine Vielzahl von Verbindungen zur Neuübertragung veranlasst werden, was den Stau verschärfen kann. Andererseits kann es auch dazu führen, daß zu viele Verbindungen abbremsen und es dadurch erst Recht zu Lastschwankungen kommt. Desweiteren könnte ein Client sich durch das Nichtverzögern des Datenverkehrs Bandbreitenvorteile verschaffen, da gehorsame Clients ihre Flüsse abbremsen, und demgemäß sind mit dem reinen RED-Algorithmus keine Quality-of-Service Konzepte umsetzbar.

Daher wurden erweiterte Algorithmen wie WRED, flow-based WRED und CHOKe entwickelt, die sich besser für die Verkehrskontrolle eignen. Das Vorgehen von **CHOKe** ist ebenfalls sehr einfach: Wenn die durchschnittliche Queuegröße  $M$  unterhalb von  $MIN$  liegt, dann wird ein ankommendes Paket  $p$  zugelassen. Liegt die durchschnittliche Queuegröße hingegen oberhalb von  $MAX$ , dann wird  $p$  abgewiesen. Der interessanteste Fall tritt für  $MIN \leq M \leq MAX$  ein: CHOKe zieht ein Paket  $q$  zufällig aus der Queue und weist das ankommende Paket  $p$  genau dann ab, wenn  $p$  und  $q$  zu demselben Fluss gehören. Offensichtlich müssen große Flüsse einen

sehr viel größeren Paketverlust erleiden als kleinere Flüsse und CHOCe sorgt mit relativ geringem Verwaltungsaufwand für die partielle Durchsetzung des Fairness-Prinzips.

Allerdings fallen „schwach aggressive“ Flüsse nicht auf: Wenn ein Router Pakete mit Rate  $n$  erhält, dann wird ein Fluß der Rate  $\sqrt{n}$  nur mit Wahrscheinlichkeit  $\frac{1}{\sqrt{n}}$  erwischt. Wie wir bereits in Satz 4.11 gesehen haben, können wir nicht erwarten, Flüsse mit größter Rate ohne großen Aufwand zu bestimmen.

TCP als verantwortliche Kontrollinstanz muss einerseits bestrebt sein, Überlast zu vermeiden und muss andererseits versuchen, die zur Verfügung stehende Kapazität der Links bestmöglich zu nutzen. Die TCP-Verkehrskontrolle misst dazu die Belastung einer Sender-Empfänger Verbindung über den bisher beobachteten Durchsatz der Verbindung und versucht, sich adaptiv auf die Belastung durch Hoch- oder Runterfahren der Emissionsrate einzustellen.

- Der Sender berechnet ein *Sendefenster* (sliding window), dessen Größe sich dynamisch dem Durchsatz anpasst, aber die Länge des Empfänger-Puffers nicht übersteigen sollte. Das Sendefenster besteht aus einem Intervall aufeinanderfolgender Pakete und beschreibt die versandbereiten Pakete, bzw. die versandten, aber noch nicht bestätigten Pakete. (Der Empfänger bestätigt den Erhalt eines Pakets durch eine *Bestätigung* – Acknowledgements oder Ack –, die Paketen in der Gegenrichtung hinzugefügt wird.) Pakete werden durchnummeriert, um dem Empfänger die Möglichkeit zu geben, die Pakete in richtiger Reihenfolge zusammenzusetzen.

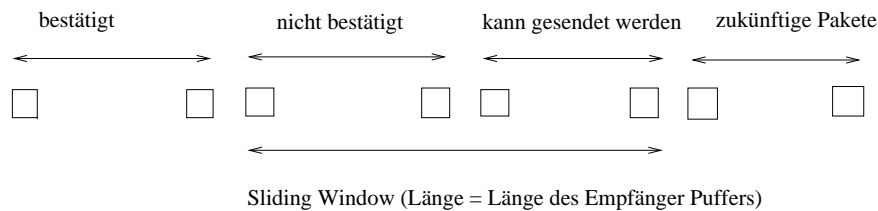


Abbildung 7.1: Sliding Window und bestätigte Pakete

- Man unterscheidet *kumulative*, *duplizierte* und *selektive* Bestätigungen. Eine kumulative Bestätigung bestätigt stets auch den Erhalt aller vorher gesandten Pakete.

Eine duplizierte Bestätigung wird verschickt, wenn ein Paket in falscher Reihenfolge empfangen wird: Nur das jüngste in richtiger Reihenfolge empfangene Paket wird bestätigt. Eine duplizierte Bestätigung ist also ein Hinweis, aber kein Beweis, für einen Paketverlust.

Eine selektive Bestätigung führt bis zu vier Paket-Intervalle explizit auf, die erfolgreich empfangen wurden. Selektive Bestätigungen sind vor Allem dann wichtig, wenn mehrere Paketverluste in einem Sendefenster auftreten und nur sie bieten die Möglichkeit, die nachzusendenden Pakete zu identifizieren.

Die verschiedenen Bestätigungsarten erklären sich einerseits aus dem nur wenigen Bytes langem Platz für Bestätigungen und andererseits aus der Redundanzanforderung, um sich auch gegen den Verlust von Bestätigungen behaupten zu können.

- Der Sender berechnet die *Rundreisezeit* (bzw. Round-Trip Time oder RTT) eines Pakets. Die Rundreisezeit ist der sich dynamisch verändernde Zeitraum, beginnend im

Versand des Pakets an den Empfänger und endend im Erhalt des vom Empfänger versandten Ack. Die Rundreisezeit wird durch Stichproben aktualisiert.

- Der *Retransmission Timeout* (RTO) ist der Zeitraum, in dem der Sender auf eine Bestätigung eines versandten Pakets wartet. Geht innerhalb des RTO-Zeitraums keine Bestätigung ein, wird das Paket wieder gesendet. Der RTO-Zeitraum verändert sich dynamisch mit der Rundreisezeit: Bei steigender Rundreisezeit, und damit bei stärker belasteten Links, wird RTO entsprechend hochgesetzt und der Sender wartet länger auf eine Bestätigung.
- Ein Fehler tritt auf, wenn der RTO-Zeitraum ohne eine Bestätigung überschritten wird (RTO-Fehler) oder wenn der Sender drei identische duplizierte Bestätigungen erhalten hat (Ordnungsfehler). (Der Sender wartet also ab in der Hoffnung, zwischenzeitlich noch eine Bestätigung zu erhalten.)
- Der Sender versucht, sein Sendefenster den Netzbedingungen anzupassen und benutzt dazu die Komponenten *Slow Start*, *Congestion Avoidance*, *Fast Retransmit* und *Fast Recovery*. Es gibt zahlreiche TCP-Varianten (TCP Tahoe, TCP Vegas, TCP Reno, TCP New Reno, TCP Sack), die sich in der Implementierung dieser vier Verfahren zum Teil wesentlich unterscheiden. Wir beschreiben TCP Sack.

(\*) Slow Start wird beim Aufbau einer Verbindung bzw. nach einem RTO-Fehler aufgerufen. (Ein RTO-Fehler wird somit als wesentlich ernster als ein Ordnungsfehler aufgefasst, denn bei einem Ordnungsfehler erhält der Empfänger ja noch neue Pakete.) Slow Start beginnt mit einem kleinen Sendefenster  $S_0$  und endet, wenn der Schwellenwert  $S_1$  ( $S_0 \ll S_1$ ) überschritten wird. Wenn Bestätigungen für alle bisher versandten Pakete innerhalb des Retransmission Timeouts eingegangen sind, dann wird die Länge des gegenwärtigen Sendefensters  $S$  auf  $S = \max\{2 \cdot S, S_1\}$  hochgesetzt.

Also steigert Slow Start die anfänglich sehr geringe Emissionsrate durch multiplikatives Anwachsen (multiplicatives increase) bis entweder der Schwellenwert  $S_1$  überschritten wird oder bis ein Fehler auftritt. Wird  $S_1$  überschritten, dann wird in das Congestion Avoidance Verfahren gewechselt.

(\*) Congestion Avoidance erhöht die Fenstergröße  $S$  um Eins (additives Anwachsen, bzw. additive increase), wann immer alle Pakete eines Fensters bestätigt wurden.

(\*) Fast Retransmit wird aufgerufen, wenn ein Ordnungsfehler für ein Paket auftritt. Das (vermutlich) verlorene Paket wird neu versandt ohne den Retransmission Timeout abzuwarten. Die gegenwärtige Emissionsrate  $S$  wird halbiert (multiplikatives Fallen, bzw. multiplicatives decrease) und Fast Recovery wird aufgerufen.

(\*) Fast Recovery: Mit Hilfe der selektiven Bestätigungen wird ein *Scoreboard* berechnet, das alle vermutlich verlorenen Pakete des Fensters aufführt. Desweiteren schätzt der Sender die Anzahl der (regulär) ausstehenden Pakete und hält die Schätzung in der Variable „pipe“ fest. Der Sender sendet Pakete des Scoreboards bzw. neue Pakete nur dann, wenn  $\text{pipe} < S$  gilt. pipe wird für jedes versandte Paket um Eins erhöht und für jede Bestätigung um Eins erniedrigt: Die Belastung des Netzes ist somit durch  $S$  beschränkt.

Tritt ein RTO-Fehler auf, wird Slow Start aufgerufen, wobei die aktuelle Fenstergröße als neuer Schwellenwert  $S_1$  gewählt wird. Werden hingegen alle Pakete des Scoreboards bestätigt, erfolgt ein Aufruf von Congestion Avoidance.

**Bemerkung 7.1** Das *Knie-Kliff* Modell erklärt die unterschiedlichen Geschwindigkeiten der einzelnen TCP-Komponenten beim Hochfahren. Dazu betrachten wir den Durchsatz als Funktion der Last und beobachten bei wachsender, aber relativ kleiner Last einen schnellen Anstieg des Durchsatzes. Ab einer gewissen Last  $L_0$  verlangsamt sich der Durchsatz merklich und kommt dann letztlich ab Last  $L_1$  völlig zum Erliegen. Wird Last  $L_1$  überschritten, kollabiert der Durchsatz fast vollständig.

Das Lastintervall  $[0, L_0]$  definiert das Knie und das Lastintervall  $[L_0, L_1]$  das Kliff. Slow Start versucht möglichst schnell, also ganz anders als sein Name besagt, die Last  $L_0$  zu erreichen. Congestion Avoidance hingegen begibt sich langsam auf den Weg zum Kliff, wohlwissend, dass der Absturz droht.

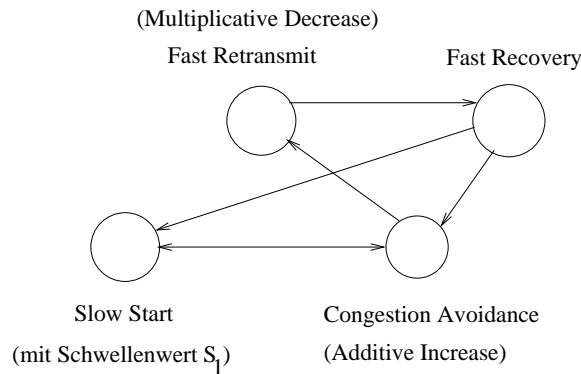


Abbildung 7.2: Interaktion von Slow Start, Congestion Avoidance, Fast Retransmit und Fast Recovery.

Die Kombination von Congestion Avoidance und Fast Retransmit folgt somit dem konservativen Prinzip des additiven Anwachsens / multiplikativen Fallens (AIMD).

**Bemerkung 7.2** TCP Tahoe, TCP Reno wie auch TCP Sack sind Implementierungen verschiedener AIMD-Verfahren. Alle Verfahren erhöhen in Congestion Avoidance ihre Senderate  $r$  additiv um  $\alpha$ , wenn kein Paket-Verlust eintritt. Die Verfahren unterscheiden sich aber in ihrer Implementierung von Fast Retransmit, also in ihrer Reaktion auf einen Paketverlust. In allerdings stark vereinfachter Darstellung ergeben sich die folgenden Definitionen. ( $\beta$  mit  $0 < \beta < 1$  und  $\gamma \in \mathbb{R}_{>0}$  sind Konstanten.)

**TCP Tahoe** (starke Geschwindigkeitsreduktion): Tritt mindestens ein Paket-Verlust auf, dann legt der Sender eine Sendepause fixierter Länge ein. Nach Ablauf der Sendepause wird die Rate  $r$  auf  $\beta \cdot r$  erniedrigt.

**TCP Reno** (mittlere Geschwindigkeitsreduktion): Tritt genau ein Paketverlust ein, dann wird die Rate auf  $\beta \cdot r - \gamma$  reduziert, aber eine Sendepause wird nicht eingelegt. Treten hingegen

mehrere Paketverluste auf, dann wird eine Sendepause von vorher festgelegter Länge eingelegt. Als neue Rate wird eine langsame, vorher fixierte Rate gewählt, die nicht von  $r$  abhängt.

**TCP Sack** (sanfte Geschwindigkeitsreduktion): Treten  $v$  Paket-Verluste auf, dann wird die Rate auf  $\beta \cdot r - \gamma \cdot v$  reduziert. Der Sender legt aber keine Sendepause ein.

In den nächsten Abschnitten versuchen wir, die Stärken und Schwächen von AIMD-Verfahren genauer zu verstehen.

## 7.1 AIMD für eine Ressource

Wir untersuchen ein stark vereinfachtes Szenario, in dem  $n$  Rechner die Ressource „Bandbreite“ nutzen möchten. Wir nehmen an, dass Rechner  $i$  zum Zeitpunkt  $t$  den Anteil  $x_i(t) \geq 0$  angefordert hat. Um die Forderung  $x_i(t+1)$  zum Zeitpunkt  $t+1$  zu bestimmen, wird  $i$  die Antwort  $y(t) \in \{\text{fahr hoch, fahr runter}\}$  des Systems abwarten und dann  $x_{t+1}$  als Funktion von  $x_i(t)$  und  $y(t)$  bestimmen. (Wir nehmen also sofortige Bestätigungen an.)

Die Aufforderung  $y(t)$  ist für alle Rechner bindend. Weiterhin fordern wir, dass alle Rechner dasselbe lineare Anforderungsschema

$$x_i(t+1) = \begin{cases} a_H + b_H \cdot x_i(t) & y_t = \text{fahr hoch,} \\ a_R + b_R \cdot x_i(t) & y_t = \text{fahr runter} \end{cases}$$

benutzen. Unser Ziel ist die Bestimmung eines linearen Anforderungsschematas, das die Fairness

$$F_{t+1} = \frac{(\sum_{i=1}^n x_i(t+1))^2}{n \cdot \sum_{i=1}^n x_i(t+1)^2}$$

zum Zeitpunkt  $t+1$  maximiert.

---

### Aufgabe 63

Zeige, dass  $0 \leq F_t \leq 1$  gilt. Darüberhinaus ist  $F_t$  genau dann maximal ( $F_t = 1$ ), wenn alle  $x_i(t)$  identisch sind.

---

**Lemma 7.1** *Ein lineares Anforderungsschema ist multiplikativ für das Runterfahren ( $a_R = 0$  und  $b_R < 1$ ) und additiv für das Hochfahren ( $b_H = 1$ ), falls den Anweisungen des Systems Folge zu leisten ist und der Fairnessanstieg in jedem Schritt zu maximieren ist.*

**Beweis:** Der Rechner  $i$  muss seine Anforderung für „ $y_t = \text{fahr runter}$ “ drosseln. In diesem Fall ist also  $x_i(t+1) = a_R + b_R \cdot x_i(t) < x_i(t)$  zu fordern, falls  $x_i(t) > 0$ . Da  $x_i(t+1) \geq 0$  auch für kleine Werte von  $x_i(t)$  sichergestellt werden muss, darf  $a_R$  nicht negativ werden. Andererseits muss  $x_i(t+1) < x_i(t)$  ebenfalls für kleine Werte von  $x_i(t)$  garantiert werden und damit folgt  $a_R = 0$ . Wir erhalten also

$$a_R = 0 \quad \text{und} \quad 0 \leq b_R < 1. \tag{7.1}$$

Für die Antwort „ $y_t = \text{fahr hoch}$ “ ist hingegen  $x_i(t+1) = a_H + b_H \cdot x_i(t) > x_i(t)$  zu fordern und wir erhalten die Bedingungen

$$a_H \geq 0 \quad \text{und} \quad b_H \geq 1. \tag{7.2}$$

Wir betrachten als Nächstes die Konvergenz gegen die optimale Fairness. Es ist

$$\begin{aligned} F_{t+1} &= \frac{(\sum_{i=1}^n x_i(t+1))^2}{n \cdot \sum_{i=1}^n x_i(t+1)^2} = \frac{(\sum_{i=1}^n a_H + b_H \cdot x_i(t))^2}{n \cdot \sum_{i=1}^n (a_H + b_H \cdot x_i(t))^2} \\ &= \frac{(\sum_{i=1}^n a_H/b_H + x_i(t))^2}{n \cdot \sum_{i=1}^n (a_H/b_H + x_i(t))^2}. \end{aligned}$$

Wir fassen den letzten Ausdruck als Funktion von  $c = a_H/b_H$  auf und erhalten die Funktion

$$G(c) = \left( \sum_{i=1}^n c + x_i(t) \right)^2 / \left( n \cdot \sum_{i=1}^n (c + x_i(t))^2 \right).$$

Beachte, dass  $G(0) = F_t$ . Wir zeigen, dass  $G'(c) \geq 0$  für  $c \geq 0$  und damit ist  $G$  für  $c \geq 0$  eine monoton wachsende Funktion. Die Bedingung  $G'(c) \geq 0$  ist äquivalent mit<sup>1</sup>

$$2n \cdot \left( \sum_{i=1}^n c + x_i(t) \right) \cdot \left( n \cdot \sum_{i=1}^n (c + x_i(t))^2 \right) \geq \left( \sum_{i=1}^n c + x_i(t) \right)^2 \cdot \left( 2n \cdot \sum_{i=1}^n (c + x_i(t)) \right)$$

und damit, nach Kürzung um den Term  $2n \cdot (\sum_{i=1}^n c + x_i(t)) > 0$  äquivalent mit

$$n \cdot \sum_{i=1}^n (c + x_i(t))^2 \geq \left( \sum_{i=1}^n c + x_i(t) \right)^2.$$

Diese Ungleichung ist stets erfüllt und der Nachweis der Monotonie von  $G$  für  $c \geq 0$  ist geführt.

Da die Funktion  $G$  für  $c \geq 0$  monoton wachsend ist, müssen wir  $c$  größtmöglich wählen, um einen maximalen Fairnessanstieg zu gewährleisten. Wenn die Anforderungen runter zu fahren sind, bleibt die Fairness damit aufgrund von (7.1) unverändert (denn  $a_R = 0$ ), während im Fall des Hochfahrens die Setzung  $b_H = 1$  wegen (7.2) optimal ist. (Allerdings ist im Fall  $a_H = 0$  zu berücksichtigen, dass die Fairness unabhängig von  $b_H$  unverändert bleibt.  $a_H = 0$  kann somit ausgeschlossen werden, da ein Fairnessanstieg für jedes  $a_H > 0$  erfolgt.)  $\square$

Das Argument zeigt, dass additives Anwachsen zu dem stärksten Anstieg der Fairness führt. Dies ist nicht weiter verwunderlich, da ein langsames Ansteigen jedem beteiligten Rechner denselben Zusatzgewinn bringt. Allerdings ist Lemma 7.1 keine wirklich überzeugende Rechtfertigung des additive increase - multiplicative decrease Rezepts, denn wir schränken uns in Lemma 7.1 freiwillig auf die kleine Klasse der linearen Anforderungsschemata ein.

---

#### Aufgabe 64

Es ist zu erwarten, dass Nutzer mit verschiedenen Bandbreiten auf die Ressource zugreifen. Diese sollen nun entsprechend ihrer Bandbreite bedient werden.

Gegeben sei eine Verteilung  $(p_i)_{1 \leq i \leq n}$  mit  $p_i \geq 0$  für jedes  $i$  und  $\sum_{i=1}^n p_i = 1$ . Steht die Ressource in Kapazität  $B$  zur Verfügung, so ist jetzt das Ziel, dass Anfrage  $i$  den Anteil  $p_i \cdot B$  erhält.

Wir definieren die gewichtete Fairness als

$$F_t = \frac{(\sum_{i=1}^n x_i(t))^2}{\sum_{i=1}^n \frac{1}{p_i} (x_i(t))^2}.$$

Beachte, dass wir die ursprüngliche Definition der Fairness für die Gleichverteilung  $p_i = \frac{1}{n}$  erhalten.

1. Zeige, dass  $0 \leq F_t \leq 1$  gilt.

---

<sup>1</sup>Zur Erinnerung:  $(\frac{f}{g})' = \frac{f' \cdot g - f \cdot g'}{g^2}$  und  $(\frac{f}{g})' \geq 0$  ist äquivalent zu  $f' \cdot g \geq f \cdot g'$ .

2. Zeige, dass  $F_t = 1 \Leftrightarrow x_j(t) = p_j \sum x_i(t)$ . **Hinweis:** Lemma 1.1 ist hilfreich. Beachte auch, dass  $\cos(\phi) = 1 \Leftrightarrow \phi = 0$  für einen Winkel  $-\pi < \phi \leq \pi$ .
3. Zeige, dass wir  $F_{t+1} \geq F_t$  für das Kommando „runter“ (beziehungsweise  $F_{t+1} > F_t$  für das Kommando „hoch“) erreichen, wenn wir

$$x_i(t+1) = \begin{cases} x_i(t) + p_i & \text{falls Kommando hoch erfolgt,} \\ \beta \cdot x_i(t) & \text{falls Kommando runter erfolgt} \end{cases}$$

wählen.

Wir verschärfen jetzt das betrachtete Szenario und lassen zu, dass alte Verbindungen terminieren und neue Verbindungen hinzukommen, bleiben aber bei der Einschränkung auf eine einzige Ressource. Wie schnell konvergiert AIMD gegen ein sich jetzt dynamisch änderndes Fairness-Optimum? Wir nehmen an, dass AIMD die Bandbreite im Hochfahren um 1 erhöht und im Runterfahren um den Faktor  $\beta < 1$  erniedrigt.

Wir fixieren eine Verbindung  $V$  und nehmen an, dass das System eine Gesamtbandbreite  $B$  zur Verfügung stellt.  $V$  wird zu einem Zeitpunkt  $T$  mit Bandbreite 0 aufgenommen und wird dann in einer Folge von Additionsschritten mehr und mehr Bandbreite gewinnen. Wir interessieren uns vor Allem für die Folge  $T_0 < T_1 < \dots < T_k < \dots$  der Zeitpunkte, zu dem das System den Befehl des Runterfahrens gibt. Es gelte  $T \leq T_0$ .

**Lemma 7.2** *Wenn  $V$  bis zum Zeitpunkt  $T_k$  besteht, dann beträgt seine Bandbreite vor dem Zeitpunkt  $T_k$  mindestens*

$$(1 - \beta)^k \cdot \frac{B}{n_k},$$

wobei  $n_k$  die Anzahl der Verbindungen zum Zeitpunkt  $T_k$  ist. Zu einem Zeitpunkt  $t \in [T_{k-1}, T_k[$  besitzt jede andere Verbindung höchstens die Bandbreite von  $V$  plus die Bandbreite  $\beta^k \cdot B$ .

**Beweis:** Wir führen den Begriff der gerechtfertigten Bandbreite für alle dem Zeitpunkt  $T$  folgenden Zeitpunkte ein. Wir legen fest, dass die von den einzelnen Verbindungen zum Zeitpunkt  $T$  gehaltenen Bandbreiten ungerechtfertigt sind. (Wir nehmen also die Sichtweise der Verbindung  $V$  ein, die zum Zeitpunkt  $T$  mit Bandbreite 0 beginnen muss.) Danach gewinnen alle Verbindungen bis zum Zeitpunkt  $T_0$  zusätzliche Bandbreite und wir legen diese gewonnene Bandbreite als gerechtfertigt fest. (Intuition: Die Verbindung  $V$  muss die ihrerseits gewonnene Bandbreite auch den anderen Verbindungen zugestehen.) Zum Zeitpunkt  $T_0$  werden alle Bandbreiten, ob gerechtfertigt oder nicht, mit dem Faktor  $\beta$  erniedrigt. Im Zeitraum  $]T_0, T_1[$  gewinnen dann alle Verbindungen gerechtfertigte Bandbreiten etc.

Wieviel Bandbreite ist zum Zeitpunkt  $T_k$  gerechtfertigt? Nur die Bandbreite zum Zeitpunkt  $T$  ist ungerechtfertigt und danach kommt nie mehr ungerechtfertigte Bandbreite hinzu. Da die ungerechtfertigte Bandbreite zu jedem Zeitpunkt  $T_i$  um den Faktor  $\beta$  erniedrigt wird, ist vor dem Zeitpunkt  $T_k$  höchstens die Bandbreite  $\beta^k \cdot B$  ungerechtfertigt.

Der Begriff der berechtigten Bandbreite ist für die Verbindung  $V$  optimal, da keine andere Verbindung mehr berechnete Bandbreite besitzen wird als  $V$ : spätere Verbindungen besitzen weniger und zwischenzeitlich terminierende Verbindungen geben zusätzliche Bandbreite frei. Das aber bedeutet, dass  $V$  vor dem Zeitpunkt  $T_k$  mindestens die Bandbreite  $(1 - \beta^k) \cdot B/n_k$  errungen hat.

Eine andere Verbindung  $W$  wird dann die meiste Bandbreite an sich reißen können, wenn sie zum Zeitpunkt  $T$  die gesamte (und damit ungerechtfertigte) Bandbreite  $B$  besitzt. Zu einem Zeitpunkt  $t \in [T_{k-1}, T_k[$  ist aber die ungerechtfertigte Bandbreite auf  $\beta^k \cdot B$  geschmolzen und

die Verbindung  $W$  besitzt dann höchstens die (gerechtfertigte) Bandbreite von  $V$  plus die ungerechtfertigte Bandbreite  $\beta^k \cdot B$ .  $\square$

Jede Verbindung erhält also mindestens ihren fairen Anteil mit exponentieller Geschwindigkeit, wenn man die Zeit in der Anzahl der Befehle zum Runterfahren misst.

Zuletzt nehmen wir das folgende, stark vereinfachte *steady-state Modell* an, um den Durchsatz  $D$  von AIMD in Abhängigkeit von der Verlustrate  $p$  zu bestimmen:

Ein Sender fährt seine Sende-Rate um jeweils ein Paket solange hoch, bis ein Schwellenwert  $S$  erreicht ist. Danach erfolgt ein Paketverlust und der Sender muss seine Rate von  $S$  auf  $\frac{S}{2}$  halbieren.

Beginnend mit Senderate  $\frac{S}{2}$  erhalten wir in den ersten  $S$  Schritten den Gesamtdurchsatz  $D^* = \sum_{i=S/2}^S i \approx \frac{3}{8} \cdot S^2$  und  $D^*$  fällt auch in jedem darauffolgendem Zeitintervall der Länge  $\frac{S}{2}$  an. Also erhalten wir den durchschnittlichen Durchsatz  $D \approx \frac{3}{4} \cdot S$ . Andererseits erfahren wir in jedem Zeitintervall genau einen Paketverlust und damit ist  $p \approx \frac{8}{3 \cdot S^2}$  die durchschnittliche Verlustrate. Als Konsequenz haben wir die Beziehung

$$D = \Theta\left(\frac{1}{\sqrt{p}}\right)$$

erhalten. Man überzeugt sich jetzt sofort, dass die Beziehung  $D = \Theta\left(\frac{1}{\sqrt{p}}\right)$  für AIAD, MIAD und MIMD gilt und AIMD besitzt die höchste Durchsatzrate als Funktion der Verlustrate.

---

#### Aufgabe 65

Wir betrachten wieder das AIMD-Verfahren für eine Ressource, die pro Zeiteinheit in Kapazität  $B$  zur Verfügung steht. Wir verwenden das Anforderungsschema

$$x_i(t+1) = \begin{cases} x_i(t) + 1 & \text{falls Kommando hoch erfolgt,} \\ \beta \cdot x_i(t) & \text{falls Kommando runter erfolgt.} \end{cases}$$

Wir nehmen an, dass die Parameter so gewählt sind, dass nach einmaligem Runterfahren die Kapazität unterschritten wird. Als Spanne eines Systems zum Zeitpunkt  $t$  bezeichnen wir die maximale Differenz  $|x_i(t) - x_j(t)|$  zwischen zwei Anfragen zum Zeitpunkt  $t$ .

- (a) Zeige, dass ein System mit  $n$  Rechnern und anfänglicher Spanne  $\Delta$  nach

$$\Theta\left(\frac{B}{n} \cdot \ln\left(\frac{\Delta}{\epsilon}\right)\right)$$

Iterationen eine vorgegebene Spanne  $\epsilon$  nicht überschreitet.

- (b) Neben der Fairness ist die Auslastung der Ressource eine wichtige Qualitätsgröße: Am gerechtesten ist schließlich ein kaputtes System. Als Auslastung  $A$  zum Zeitpunkt  $t$  definieren wir

$$A(t) = \begin{cases} \sum_{i=1}^n x_i(t)/B & \text{falls } \sum_{i=1}^n x_i(t) \leq B, \\ A(t-1) & \text{sonst.} \end{cases}$$

Wir beschränken uns auf einen Rechner ( $n = 1$ ). Zeige, dass der mittlere Auslastungsfaktor  $\frac{1}{t} \sum_{t'=1}^t A(t')$  mit wachsendem  $t$  gegen einen Wert von mindestens  $\frac{1}{2} \cdot (\beta + 1)$  strebt.

---

Man nennt einen Paketfluss mit Durchsatz  $D = \Theta\left(\frac{1}{\sqrt{p}}\right)$  auch *TCP-freundlich*, da der Fluss zumindest scheinbar seine Geschwindigkeit nach den TCP-Regeln steuert. Da ein Router die Beziehung Durchsatz/Verlustrate nachprüfen kann, können somit aggressive Flüsse, erkannt und eliminiert werden.

---

**Aufgabe 66**

Die TCP-Geschwindigkeitsregeln sind für Streaming Audio- und Video-Applikationen zu konservativ und man betrachtet deshalb auch die folgenden Geschwindigkeitsregeln: Beim Hochfahren wird  $x_i(t+1) = x_i(t) + \alpha \cdot x_i(t)^k$  und beim Herunterfahren wird  $x_i(t+1) = x_i(t) - \beta \cdot x_i(t)^l$  für  $\alpha, \beta > 0$  gefordert.

(a) Wir nehmen an, dass sich zwei Rechner um eine Ressource streiten und dass sich diese beiden Rechner nach den neuen Geschwindigkeitsregeln richten. Wir nehmen desweiteren an, dass das System genau dann auf die Bremse tritt, wenn  $x_1(t) + x_2(t) > 1$ . Zeige, dass  $\lim_{t \rightarrow \infty} (x_1(t), x_2(t)) = (1/2, 1/2)$  gilt.

(b) Verallgemeinere Teil (a) auf den Fall von beliebig vielen Rechnern.

(c) Zeige, dass  $D = \Theta(\frac{1}{p^{1/(k+l+1)}})$  gilt. Also erzeugen die neuen Geschwindigkeitsregeln genau dann TCP-freundliche Flüsse, wenn  $k+l=1$  (und  $l \leq 1$ ) gilt.

**Offenes Problem 3**

Entwerfe ein realistisches Szenario, in dem AIMD als fast-optimal nachgewiesen werden kann. In [EDD] wird ein On-line Szenario mit einer einzelnen Ressource entworfen.

## 7.2 Alternative Geschwindigkeitsregeln

Zwar setzt sich AIMD in der obigen Analys gegen MIMD, AIAD und MIAD durch, doch ist ein Vergleich mit einer größeren Klasse von Strategien unterblieben und nur der Fall einer einzigen umkämpften Ressource wurde betrachtet. Schließlich haben wir unter idealisierten Annahmen wie einer nicht zu berücksichtigenden Round-Trip Time gearbeitet. Die folgenden Kritikpunkte müssen deshalb angesprochen werden:

- (1) TCP ändert Geschwindigkeiten nur aufgrund von Paket-Verlusten.

Der Paket-Verlust ist nicht anderes als ein binäres Signal, das aber nicht den Grad der Verstopfung und damit keine Möglichkeit einer der Ursache angepassten Reaktion erlaubt. Da das Feststellen eines Paketverlusts ohne Zusatzinformation erfolgt, ist die Reaktion notwendigerweise defensiv und Durchsatz-feindlich. Vorbeugende Gefahrmeldungen durch Router scheinen vernünftiger als ein Warten auf den Unfall.

- (2) Die Reaktionszeit hängt stark von der Round-Trip Time ab, die kontroll-theoretisch als Feedback-Verzögerung wirkt.

Als Konsequenz benachteiligt TCP Paketflüsse mit grosser Round-Trip Time. Um Stabilität bei großer Feedback-Verzögerung zu bewahren, müssen Sender zwangsläufig entsprechend langsamer reagieren, aber das Ausmaß der Geschwindigkeitsveränderung ist unklar: Wenn Sende- und Empfangsrate auseinanderklaffen, dann ist aus Sicht der Kontrolltheorie ein invers proportionales Abbremsen angemessen. Wenn hingegen auf eine durch eine Queue verursachte Verzögerung reagiert werden muss, dann ist ein invers quadratisches Abbremsen angeraten.

- (3) TCP skaliert schlecht mit steigender Link-Kapazität von optischen Links und wachsender Verzögerung auf Satelliten-Links.

Sally Floyd berechnet, dass eine standard TCP-Verbindung mit Paketen von 1.5 KB, einer Round-Trip Time von 100 Millisekunden und einem Steady-State Durchsatz von 10 Gigabit pro Sekunde ein Sliding Window von ungefähr 84,000 Segmenten und eine Verlustrate von Eins zu 5 Milliarden (oder äquivalent ein verlorenes Paket in 1 2/3 Stunden) benötigt. Solche Anforderungen sind durch das konventionelle TCP nicht zu bewältigen.

- (4) Slow Start benötigt  $\Theta(\log_2 N)$  Verdopplungen bis die volle Bandbreite  $N$  erreicht ist.

Dies ist kein Problem für langlebige Flüsse, aber sehr wohl ein Problem für kurzlebige Flüsse, die während ihrer Lebenszeit Slow Start nicht verlassen und dementsprechend die zur Verfügung stehende Bandbreite auch nicht voll nutzen können. (Beachte, dass ein Zeitfenster der Länge mindestens RTT zwischen Verdopplungen liegen muss.)

Die zukünftige Form der Verkehrskontrolle ist unklar. Zum jetzigen Zeitpunkt konkurrieren die folgenden Modifikationsvorschläge miteinander.

**HighSpeed TCP** ([ftp.rfc-editor.org/in-notes/rfc3649.txt](http://ftp.rfc-editor.org/in-notes/rfc3649.txt)) verhält sich wie das konventionelle TCP, wenn das Sliding Window nicht zu groß ist. Auch für große Sliding Windows wird das AIMD Verfahren beibehalten, allerdings mit aggressiveren Inkrementen und langsamerem Abbremsen.

Desweiteren wird Slow Start durch Limited Slow Start ersetzt, wenn TCP mit einem großen Sliding Window beginnt: Statt die Senderate exponentiell zu steigern, wird die Fenstergröße nach jedem Acknowledgement um eine Konstante vergrößert. Selbst wenn ein Acknowledgement ausbleibt, wird die Fenstergröße allerdings um eine kleinere Konstante vergrößert.

**QuickStart** ([www.ietf.org/internet-drafts/draft-amit-quick-start-04.txt](http://www.ietf.org/internet-drafts/draft-amit-quick-start-04.txt))

Bei Eröffnung einer Verbindung zwischen Hosts  $A$  und  $B$  sendet  $A$  ein Paket mit einer gewünschten Emissionsrate. Router auf dem Weg von  $A$  nach  $B$  können die gewünschte Emissionsrate billigen, modifizieren oder ablehnen. Host  $B$  sendet dann das Ergebnis der "Router-Umfrage" an  $A$  zurück. Sollte die Anfrage abgelehnt werden, wählt  $A$  eine Default-Emissionsrate. Natürlich ist das Ziel, bereits mit einer großen Emissionsrate, beziehungsweise mit einem großen Sliding Window zu beginnen.

Beide Verfahren sind also konservative TCP-Erweiterungen und die Kritikpunkte (1) und (2) bleiben bestehen.

**Explicit Control Protocol (XCP)** geht auf die Publikation [KHR] zurück. Siehe auch <http://www.isi.edu/isi-xcp/> und <http://www.ana.lcs.mit.edu/dina/XCP/>. XCP verfolgt zwei neue Ansätze. Zuerst verschickt XCP *explizite* Stauinformationen: Die Router informieren den Sender durch Acknowledgements über den Verstopfungsgrad und erlauben damit eine der Ursache angepasste Reaktion. Zuletzt behandeln Router *Effizienz* (möglichst große Ausnutzung der Bandbreite) und *Fairness* (möglichst gleichmäßige Aufteilung der Bandbreite) getrennt, was zu einer schnelleren, aber immer noch fairen Bandbreitennutzung führen soll.

In XCP gibt der Sender  $A$  die Größe seines Sliding Windows und die gegenwärtige Round-Trip Time im Paket-Header bekannt und nennt anfänglich seine gewünschte Emissionsrate  $S$ . Jeder Router auf dem Weg modifiziert  $S$  aufgrund dieser Information, schreibt die modifizierte Information in den Paket-Header und reicht das Paket an den nächsten Router weiter. Der Empfänger  $B$  verschickt dann die Empfehlung als Teil des Acknowledgements zurück an  $A$ . Insbesondere kann somit QuickStart als Teil von XCP aufgefasst werden.

Jeder Router  $R$  bestimmt für jeden Link die durchschnittliche Round-Trip Time  $T$  der über den Link führenden Pakete.  $R$  beginnt jetzt ein Zeitintervall der Länge  $T$  mit einer Geschwindigkeitsempfehlung und wartet  $R$  die nächsten  $T$  Schritte ab, um die Konsequenz der bisherigen Empfehlung einschätzen zu können. Danach wird ein neues Zeitfenster mit aktualisiertem  $T$  aufgemacht und das Vorgehen wiederholt sich.

Die Geschwindigkeitsempfehlung wird vom *Effizienz-Controller* und dem *Fairness-Controller* entworfen. Der Effizienz-Controller bestimmt die freie Bandbreite  $B$  am Ende des alten Zeitfensters sowie (im Wesentlichen) die minimale Queuegröße  $Q$  während des letzten Zeitfensters und definiert dann

$$\phi = \alpha \cdot B - \beta \cdot Q$$

als zu vergebende, bzw. zu drosselnde Bandbreite für Koeffizienten  $\alpha, \beta \in [0, 1]$ . (Die minimale Queuegröße geht negativ ein, damit die Queue entleert werden kann.) Der Effizienz-Controller bearbeitet somit den gesamten eingehenden Verkehr nach dem MIMD-Prinzip. Der Fairness-Controller wendet AIMD an, um  $\phi$  auf die Flüsse aufzuteilen:

- Für  $\phi \geq 0$  erhält jeder Fluss dasselbe additive Inkrement.
- Für  $\phi < 0$  sinkt die Bandbreite eines jeden Flusses um denselben Faktor.

Beachte, dass für  $\phi = 0$  die Effizienz maximiert wurde und damit ein weiterer Fairnessanstieg ausgeschlossen ist. Deshalb wird die Bandbreite  $\max\{0, \frac{b}{10} - |\phi|\}$  des Gesamtflusses zur Fairness-Umverteilung benutzt ( $b$  ist die Bandbreite des Gesamtflusses).

Schließlich müssen wir noch den Fehlerfall ansprechen. Hier verhält sich XCP wie TCP. XCP ist sogar fairer als das konventionelle TCP, da in TCP eine Umverteilung nur im Fehlerfall geschieht. Desweiteren kann gezeigt werden, dass XCP in einem vereinfachten Modell (eine strittige Ressource mit  $n$  Rechnern, die die gleiche Round-Trip Time besitzen) gegen ein Fairness und Effizienz Optimum konvergiert.

Experimentelle Untersuchungen [KHR] zeigen, dass XCP eine höhere Bandbreitenausnutzung, kleinere Queuegrößen und eine geringere Verlustrate besitzt. Gleichzeitig fordert XCP keinen großen administrativen Aufwand und sollte von Hochgeschwindigkeitsroutern benutzbar sein. Allerdings bleibt die Frage, ob ein Protokollwechsel durchsetzbar ist: Siehe auch



## Kapitel 8

# IP-Traceback

In einer typischen Verbindung sendet ein Benutzer eine Anfrage an einen Web-Server mit der Bitte um Zugang. Der Web-Server beantwortet die Anfrage positiv, der Benutzer bestätigt Erhalt des Zugangsrechts und erhält Zugriff. In einer Denial-of-Service Attacke schickt ein Angreifer eine Vielzahl von Zugangsanfragen mit falscher Rückadresse an sein Opfer. Der Server erhält somit keine Bestätigung, wartet aber manchmal bis zu einer Minute bevor die Verbindung gekappt wird. Dieser Prozess wiederholt sich und die Website wird lahmgelegt. Im allgemeinen Fall wird ein Web-Server mit Anfragen (E-Mail, IP-Ping Anfragen, Internet Control Message Protocol (ICMP) Pakete) überschwemmt und die Ressourcen des Servers werden aufgebraucht. Ein solcher Angriff geht im „besten“ Fall von einem einzigen Rechner aus, kann aber auch eine Vielzahl gehackter Rechner in einer verteilten Denial-of-Service Attacke miteinbeziehen.

Wir stellen uns das Problem, den Weg der angreifenden Pakete zu dem ersten gehackten Rechner zurückzuverfolgen. Man beachte, dass der Paket-Header nur die IP-Adresse des Senders und des Empfängers bereitstellt und neben dem Hop-Count keine weitere Information über den zurückgelegten Weg zur Verfügung stellt. Leider lässt das IP-Protokoll zu, dass der Sender seine IP-Adresse eintragen darf und gefälschte IP-Adressen sind somit leicht zu erstellen.

Wir stellen zuerst eine Reihe tatsächlicher benutzter Methoden sowie potentielle Methoden mit ihren Vor- und Nachteilen vor.

- Anhängen von ID's der durchlaufenen Router: Dieses sicherste Verfahren scheitert an der vorgegebenen Maximallänge des Paket-Headers. Desweiteren ist es schwierig eine Maximallänge für den Weg vorzugeben, da der Angreifer ein gefälschtes Anfangsstück des Weges einsetzen kann und damit den vorgesehen Platz füllt.
- Filterung: Router bestimmen, ob die Adresse des Senders legal ist. Ein solcher Ansatz ist für geringes Paketvolumen und kleine Netze adäquat, da dann die Legalität überprüfbar ist. Allerdings besteht die Gefahr, dass eine legale Adresse gefälscht wird.
- Testen der Links: Der am nächsten zum Opfer liegende Router rekonstruiert die Links, auf denen der Angriffsverkehr verläuft. Dieses Vorgehen setzt voraus, dass der Angriff andauert, bzw. dass der Angreifer auf die Rückverfolgung nicht reagieren kann. Desweiteren wird die Rückverfolgung den Routern aufgeladen.
- Logging: Erhaltene und weitergeleitete Pakete werden in einem „Logbuch“ festgehalten. Dieser Ansatz verbraucht enorme Ressourcen und setzt voraus, dass verschiedenste Provider in der Rückverfolgung kooperieren.

- ICMP Traceback<sup>1</sup>: Es wird eine Stichprobe der erhaltenen Pakete angelegt (mit einer Stichprobenwahrscheinlichkeit von z.B. 1/20000). Sodann wird für jedes „gezogene“ Paket eine ICMP-Traceback Nachricht an den Empfänger verschickt, wobei die Nachricht die Router auf dem Weg zum Empfänger aufführt. Dieser Zugang ist erfolgversprechend, allerdings wird die geringe Stichprobengröße eine Rekonstruktion behindern und auch der Angreifer erhält die Möglichkeit gefälschte ICMP-Nachrichten abzuschicken.

**Das Modell:** Wir repräsentieren das Opfer als Wurzel eines Baums  $B$  und Router als innere Knoten von  $B$ . Die inneren Knoten entsprechen potentiellen Angreifern. Unser Ziel ist die Rekonstruktion der Angriffswege. Wir machen die folgenden Annahmen:

- Ein oder mehrere Angreifer attackieren das Opfer, wobei Angreifer zusammenarbeiten und über Rekonstruktionsversuche im Bilde sind.
- Pakete können verloren gehen, bzw in anderer Reihenfolge empfangen werden.
- Router müssen ihre Berechnungskraft vorwiegend für den Datentransport einsetzen.
- Angriffe sind nur ernst zu nehmen, wenn sie das Opfer wesentlich behindern. Insbesondere sind Tausende bis zu Millionen angreifender Pakete erforderlich.
- Innerhalb kurzer Zeit von einem Angreifer abgeschickte Pakete benutzen im Wesentlichen denselben Weg.

Wir stellen einen ersten Verteidigungsmechanismus vor, der aus einer randomisierten Markierung der Pakete durch die Router des Weges sowie aus einem Rekonstruktionsalgorithmus des Opfers besteht. Die einzig notwendige Kooperation unter den Routern besteht somit in der Paket-Markierung und die Rekonstruktion wird allein vom Opfer durchgeführt. Wir müssen allerdings annehmen, dass der Paket-Header drei zusätzliche Felder zur Verfügung stellt: ein Start- und Endfeld sowie ein Distanzfeld, die insgesamt bis zu 72 Bits (zweimal 32 Bits für die IP-Adressen und 8 Bits für das Distanzfeld) verbrauchen. (Die beiden Start- und Endfelder werden die Endknoten einer durchlaufenen Kante festhalten, das Distanzfeld hält den Abstand zwischen dem Endknoten und dem gegenwärtigen Knoten fest.)

### Algorithmus 8.1 Randomisierte Markierung

Wenn ein Router einen zu hohen Distanzwert entdeckt, dann wird er sich im Startfeld eintragen und das Distanzfeld auf Null setzen. Ansonsten entscheidet sich ein Router  $R$  mit Wahrscheinlichkeit  $p$ , ein Paket  $P$  zu markieren. Wenn  $R$  das Paket markiert, dann trägt er sich im Startfeld ein und setzt das Distanzfeld auf Null.

Wenn  $R$  das Paket nicht markiert und das Distanzfeld den Wert Null hat, dann trägt sich  $R$  in das Endfeld ein. Jeder nicht markierende Router wird das Distanzfeld inkrementieren.

*Kommentar:* Ein Angreifer besitzt zwar die Möglichkeit, seine Pakete zu markieren, aber die meisten seiner Pakete werden von Upstream-Routern überschrieben. Selbst wenn ein vom Angreifer markiertes Paket durchkommt, dann erzwingt die Distanz-Markierung, dass die tatsächliche Distanz zum Angreifer mindestens so groß ist wie die im Paket

---

<sup>1</sup>ICMP (Internet Control Message Protocol) ist eine Komponente des Internet Layers. Es benutzt das Internet Layer, um Probleme im Paketversand bekannt zu geben.

vermerkte Distanz: Es wird dem Angreifer nicht gelingen, den Verdacht auf eine Kante zwischen ihm und dem Opfer zu lenken, aber es ist durchaus möglich, den Verdacht auf weiter entfernt liegende Kanten zu lenken.

### Algorithmus 8.2 Rekonstruktion durch das Opfer

- (1) Das Opfer initialisiert einen Baum und trägt sich als Wurzel ein.
- (2) Wenn ein angreifendes Paket mit der Markierung (start, ende, distanz) empfangen wird, dann setzt das Opfer die Kante (start, ende) ein, falls noch nicht vorhanden und falls die vermerkte Distanz mit der tatsächlichen Distanz zwischen ende und Opfer übereinstimmt.

*Kommentar:* Wenn die vermerkte Distanz mit der tatsächlichen Distanz zwischen ende und Opfer übereinstimmt, dann ist der Angreifer ein Nachfahre des Knotens ende.

- (3) Die längsten Wege des Baums zur Wurzel entsprechen den vermuteten Angriffswegen.

Wir stellen die wesentlichen Eigenschaften der beiden Algorithmen zusammen.

**Satz 8.3 (a)** *Wenn sich ein Angreifer im Abstand  $d$  vom Opfer befindet, dann wird sein Weg nach einer erwarteten Anzahl von höchstens*

$$\frac{\ln(d) + O(1)}{p \cdot (1-p)^{d-1}}$$

*Paketen offengelegt. (Wenn  $p \approx \frac{1}{d}$ , dann genügen somit  $O(d \cdot \ln(d))$  Pakete. Die Wahrscheinlichkeit  $p = \frac{1}{25}$  wird empfohlen, da Wege nur selten die Länge 25 überschreiten.)*

**(b)** *Wenn  $\alpha$  Angreifer teilnehmen, dann erhöht sich die für die Rekonstruktion aller Wege notwendige erwartete Paketzahl um  $\alpha$ .*

**Beweis (a):** Die Wahrscheinlichkeit, dass das Opfer ein vorgegebenes Paket erhält, das zuletzt vom  $i$ ten Router des Weges markiert wurde, ist genau  $p \cdot (1-p)^{d-i}$ . Also ist die Wahrscheinlichkeit, dass irgendein Knoten des Weges ein vorgegebenes Paket markiert, mindestens  $\sum_{i=1}^d p \cdot (1-p)^{d-i} \geq d \cdot p \cdot (1-p)^{d-1}$ .

Wir wissen andererseits aus dem Coupon-Collector Problem, dass eine erwartete Anzahl von  $d \cdot \ln(d) + O(d)$  Versuchen genügt, um sämtliche Coupons zu erhalten. Also genügt die erwartete Paket-Zahl

$$\frac{d \cdot \ln(d) + O(d)}{d \cdot p \cdot (1-p)^{d-1}} = \frac{\ln(d) + O(1)}{p \cdot (1-p)^{d-1}}$$

für die Rekonstruktion. **(b)** ist offensichtlich. □

---

#### Aufgabe 67

Wir untersuchen in dieser Aufgabe die Relevanz einer guten Wahl der Markierungswahrscheinlichkeit  $p$  beim Probabilistic Packet Marking. Wir nehmen an, dass ein Angreifer Pakete entlang eines Weges der Länge  $d$  sendet. Das Ziel ist eine Rekonstruktion des kompletten Weges. Sei  $W(d)$  die Wartezeit bis der komplette Weg bekannt ist.

Wir wissen bereits, dass  $E[W(d)] = O(d \cdot \log(d))$  für  $p = \frac{1}{d}$  gilt.

(a) Zeige:  $E[W(d)] = \Omega(d^k \cdot \log(d))$  für  $p = \frac{1}{d^k}$  und  $k > 1$ .

(b) Zeige:  $E[W(d)] = \omega(d^k)$  für  $p = \frac{1}{\sqrt[k]{d}}$  und beliebiges festes  $k$ .

Ein Schwachpunkt dieses Verfahrens ist die große Anzahl der in Anspruch genommenen Bits im Paket-Header. Wir stellen deshalb einige Varianten vor, die die Anzahl benötigter Bits signifikant reduzieren.

Eine erste Modifikation benötigt statt den beiden Start- und Endefeldern nur ein einziges Kanten-ID-Feld von 32 Bits und halbiert somit fast die benötigte Bitzahl: Wenn ein Router  $R$  sich entscheidet, ein Paket zu markieren, dann wird der nächste Router  $R'$  des Weges das XOR  $R \oplus R'$  berechnen. Wenn das Opfer über den Weg  $(R_1, \dots, R_k)$  angegriffen wird, dann wird es mit  $R_k, R_{k-1} \oplus R_k, \dots, R_i \oplus R_{i+1}, \dots$  markierte Pakete erhalten, die es dann nacheinander unter Benutzung des Distanzfeldes dekodieren kann.

---

#### Aufgabe 68

Wir versuchen den Speicheraufwand von 72 (bzw. 40) Bits weiter zu reduzieren.

- (a) Zeige, wie man durch Aufteilung der Adressen in  $k$  Fragmente den Speicheraufwand weiter reduzieren kann. Welche Probleme können hierbei auftreten (wenn Pakete von mehr als einem Angreifer analysiert werden) und wie kann man diese Probleme umgehen? Welcher Speicherbedarf ist mit dieser Methode erreichbar?

Wie läuft die Kodierung und Dekodierung der Weginformation ab, wenn das Opfer, ausgehend von einem einzelnen Angreifer, den Weg vom ersten Router nach dem Angreifer bis zum Opfer rekonstruieren will?

- (b) Zeige, dass die erwartete Anzahl eintreffender Pakete (die benötigt werden, um den Weg zu rekonstruieren) bei einer Weglänge  $l$  und einer Markierungswahrscheinlichkeit  $p$  durch

$$\frac{k \cdot \ln(k \cdot l)}{p(1-p)^{l-1}}$$

nach oben beschränkt ist.

---

Eine substantielle Verbesserung kann nicht erreicht werden, wenn nur die Typen verschiedener Nachrichten, nicht aber ihre Anzahl festgehalten wird:

**Lemma 8.4** *Das Opfer sei die Wurzel eines vollständigen binären Baums der Tiefe  $n$  und ein Angreifer befinde sich auf einem unbekanntem Blatt des Baums. Jedes Paket besitze ein Markierungsfeld von  $b$  Bits, das durch die Router des durchlaufenen Wegs beschreibbar ist. Der Inhalt des Markierungsfelds definiere den Typ des Pakets.*

*Wenn das Opfer nur die Typen der erhaltenen angreifenden Pakete festhält und nicht deren jeweilige Anzahl, dann muss  $b \geq \log_2(n-1)$  gelten, damit das Opfer das angreifende Blatt mit Wahrscheinlichkeit mindestens  $\frac{1}{2}$  rekonstruieren kann.*

**Beweis:** Es gibt  $2^b$  verschiedene Pakettypen und  $2^{2^b}$  verschiedene Teilmengen von Pakettypen. Das Opfer muss aus der Teilmenge der erhaltenen Pakettypen auf das angreifende Blatt zurückschließen können. Wenn dieser Rückschluss auf eines der  $2^n$  Blätter mit Wahrscheinlichkeit mindestens  $\frac{1}{2}$  gelingen soll, dann muss die Anzahl dieser Teilmengen mindestens  $2^n/2$  betragen und

$$2^{2^b} \geq \frac{2^n}{2}$$

folgt. Nach zweimaligem Logarithmieren erhalten wir  $b \geq \log_2(n-1)$ . □

Wir zeigen jetzt, dass das Zählen der erhaltenen Nachrichten pro Typ einen entscheidenden Vorteil bringt. Allerdings wird dieses Schema nur für *ein* angreifendes Blatt erfolgreich sein und wir müssen auf das Markierungsschema von Algorithmus 8.1 für den allgemeinen Fall zurückgreifen.

Wir nehmen der Einfachheit halber an, dass das Opfer der Wurzel eines binären Baums der Tiefe  $n$  entspricht. (Diese Annahme kann entfernt werden und dient nur einer einfacheren Beschreibung.) Unser Ziel ist die Rekonstruktion des angreifenden Blatts mit einem einzigen Bit! Da wir in einem binären Baum arbeiten, dessen Kanten wir uns Null/Eins markiert vorstellen, wird der Angreifer durch einen unbekannt binären String  $B = (B_1, \dots, B_n)$  beschrieben. Unser neues Markierungsschema wird die Wahrscheinlichkeit  $p$ , dass das Opfer ein mit Eins markiertes angreifendes Paket erhält, so steuern, dass vom Wert von  $p$  auf den Bitstring  $B$  zurückgeschlossen werden kann. Ein solcher Rückschluss ist zum Beispiel dann trivial, wenn  $p = \sum_{i=1}^n B_i \cdot 2^{-i}$ . Für die Dekodierung eines ähnlichen Schemas benötigen wir die folgende Beobachtung:

**Lemma 8.5** *Unbekannte Bits  $B_1, \dots, B_n$  sind vorgegeben sowie bekannte reelle Zahlen  $p, \sigma$  und  $c_1, \dots, c_n$  mit den folgenden Eigenschaften:*

- (1)  $|p - \sum_{i=1}^n c_i \cdot B_i| \leq \sigma$ ,
- (2) für jedes  $i$  ( $1 \leq i \leq n-1$ ) gilt  $c_i > 2 \cdot \sigma + \sum_{j=i+1}^n c_j$  und
- (3)  $c_n > 2 \cdot \sigma$ .

Dann kann die Bitfolge  $(B_1, \dots, B_n)$  aus der Folge  $(p, \sigma, c)$  rekonstruiert werden.

**Beweis:** Es genügt zu zeigen, dass Bit  $B_i$  rekonstruiert werden kann, wenn  $B_1, \dots, B_{i-1}$  bereits rekonstruiert wurden. Wir behaupten, dass  $c_{n-k} > 2^{k+1} \cdot \sigma$  als Konsequenz der Eigenschaften (2) und (3) folgt. Wir führen ein induktives Argument und stellen zuerst  $c_n > 2^1 \cdot \sigma$  mit Eigenschaft (3) fest. Der Induktionsschluss folgt unmittelbar, da mit Eigenschaft (2)

$$c_{n-k} > 2 \cdot \sigma + \sum_{j=0}^{k-1} c_{n-j} > 2 \cdot \sigma + \sum_{j=0}^{k-1} 2^{j+1} \cdot \sigma = 2^{k+1} \cdot \sigma$$

gilt. Insbesondere ist also stets  $c_i > 0$ .

**Fall 1:**  $p - \sum_{j=1}^{i-1} c_j \cdot B_j \geq c_i - \sigma$ . Wir wenden wieder Eigenschaft (2) an und erhalten  $p - \sum_{j=1}^{i-1} c_j \cdot B_j \geq c_i - \sigma > \sigma + \sum_{j=i+1}^n c_j$ . Also folgt

$$p - \sum_{j=1}^{i-1} c_j \cdot B_j - \sum_{j=i+1}^n c_j \cdot B_j \geq p - \sum_{j=1}^{i-1} c_j \cdot B_j - \sum_{j=i+1}^n c_j > \sigma.$$

Da Eigenschaft (1) für  $B_i = 0$  verletzt wird, ist zwangsläufig  $B_i = 1$ .

**Fall 2:**  $p - \sum_{j=1}^{i-1} c_j \cdot B_j < c_i - \sigma$ . Diesmal ist

$$p - \sum_{j=1}^{i-1} c_j \cdot B_j - c_i \cdot B_i - \sum_{j=i+1}^n c_j \cdot B_j \leq p - \sum_{j=1}^{i-1} c_j \cdot B_j - c_i \cdot B_i$$

und Eigenschaft (1) wird für  $B_i = 1$  verletzt: Es ist zwangsläufig  $B_i = 0$ . □

Sei  $r = \frac{1}{2} - \varepsilon$  für ein später zu bestimmendes  $\varepsilon$ . Wir entwerfen ein Markierungsschema, das  $\sum_{i=1}^n B_i \cdot r^{-i} / 2$  als Wahrscheinlichkeit einer vom Opfer erhaltenen Eins erzwingt. Sei  $R_n$  das angreifende Blatt und sei  $R_i$  der  $i$ te Router auf dem Weg zur Wurzel  $R_0$ .

**Algorithmus 8.6 Randomisierte Markierung mit einem Bit**

$R_i$  erhält ein Paket von Router  $R_{i+1}$ . Das im Paket enthaltene Bit sei  $C_i$ . Der Router  $R_i$  bestimmt das Bit  $B_i$ , das die Kante zum Kind  $R_{i+1}$  beschriftet.

Wenn  $(B_i, C_i) = (0, 0)$ , dann wird das Bit  $C_i$  mit Null überschrieben. Wenn  $(B_i, C_i) = (0, 1)$  (bzw.  $(B_i, C_i) = (1, 0)$  oder  $(B_i, C_i) = (1, 1)$ ), dann wird  $C_i$  mit Wahrscheinlichkeit  $r$  (bzw.  $\frac{1}{2}$  oder  $1 - \varepsilon$ ) durch eine Eins ersetzt. Ist der Münzwurf nicht erfolgreich, dann wird  $C_i$  durch eine Null ersetzt. Das modifizierte Paket wird an  $R_{i-1}$  weitergeleitet.

Wir überprüfen zuerst, dass das Opfer auch tatsächlich eine Eins mit der gewünschten Wahrscheinlichkeit  $\sum_{i=1}^n B_i \cdot r^{i-1}/2$  erhält. Für  $t \in \{0, 1\}$  sei  $p_i^t$  die Wahrscheinlichkeit, dass Router  $R_i$  das Bit  $C_i = 1$  erhält, wenn der Angreifer zu Anfang das Paket mit dem Bit  $t$  markiert hat.

Wenn  $B_i = 0$ , dann ist  $p_{i-1}^t = r \cdot p_i^t$ . Für  $B_i = 1$  hingegen ist  $p_{i-1}^t = \frac{1}{2} \cdot (1 - p_i^t) + (1 - \varepsilon) \cdot p_i^t = \frac{1}{2} + (\frac{1}{2} - \varepsilon) \cdot p_i^t = \frac{1}{2} + r \cdot p_i^t$ . Also ist

$$p_{i-1}^t = r \cdot p_i^t + B_i \cdot \frac{1}{2}$$

und wir erhalten

$$p_0^t = t \cdot r^n + \sum_{i=1}^n B_i \cdot r^{i-1}/2$$

durch Induktion. Für die Analyse von Algorithmus 8.6 müssen wir weiterhin sicherstellen, dass wir bei hinreichend großer Paketzahl  $P$  hochwahrscheinlich eine scharfe Approximation  $p$  von  $\sum_{i=1}^n B_i \cdot r^{i-1}/2 = p_0^t - t \cdot r^n = p_0^0$  erhalten. Wenn  $x$  die Anzahl der Pakete mit Bit 1 ist, dann geben wir  $p = \frac{x}{P} - r^n/2$  als Schätzung an.

Wir haben uns damit, unabhängig von der Wahl des vom Angreifer eingesetzten Bits, bereits den Fehler  $r^n/2$  eingehandelt. Dieser Verlust ist unvermeidlich, denn der erwartete Anteil  $p^*$  der Einsen liegt im Intervall  $[\sum_{i=1}^n B_i \cdot r^{i-1}/2, r^n + \sum_{i=1}^n B_i \cdot r^{i-1}/2] = [p_0^0, p_0^1]$  und der Angreifer kontrolliert den exakten Wert. Wir machen den Ansatz

$$P = 3 \cdot \ln(2/\delta)/(\varepsilon^2 \cdot r^{2 \cdot n})$$

und untersuchen die Wahrscheinlichkeit, dass  $|p - p^*| > \frac{r^n}{2} + \varepsilon \cdot r^n$ . Die Anwendung einer Variante der Chernoff-Schranke liefert dann die Schranke

$$\text{prob}[|p - p^*| > \frac{r^n}{2} + \varepsilon \cdot r^n] \leq \delta.$$

Wir dekodieren mit Hilfe von Lemma 8.5 und zeigen zuerst, dass die Eigenschaften von Lemma 8.5 für  $p$ ,  $\sigma = \frac{r^n}{2} + \varepsilon \cdot r^n$  und  $c_i = r^{i-1}/2$  mit Wahrscheinlichkeit mindestens  $1 - \delta$  erfüllt sind. Eigenschaft (1) ist eine Konsequenz der Chernoff-Schranke. Für Eigenschaft (3) müssen wir  $\frac{r^n}{2} + \varepsilon \cdot r^n < c_n/2 = r^{n-1}/4$  fordern. Diese Forderung ist äquivalent zu der Ungleichung  $r \cdot (\frac{1}{2} + \varepsilon) < 1/4$ , die wegen  $r = \frac{1}{2} - \varepsilon$  erfüllt ist. Eigenschaft (2) folgt aus Eigenschaft (3), da  $c_i - \sum_{j=i+1}^n c_j > c_n$ .

**Satz 8.7** *Algorithmus 8.6 erlaubt die Rekonstruktion eines Weges, wenn mindestens  $P = 3 \cdot \ln(2/\delta)/(\varepsilon^2 \cdot r^{2 \cdot n})$  Pakete vom Opfer erhalten werden.*

Die Anzahl der Pakete ist von der Form  $O((2 + \varepsilon')^{2 \cdot n})$  und damit zu groß für praktische Anwendungen. Allerdings kann unser Schema für  $b$  Bits so modifiziert werden, dass die notwendige erwartete Paketzahl auf ungefähr  $O((2 + \varepsilon')^{4 \cdot n/2^b})$  gedrückt wird. Aber der etwas verschwenderische Algorithmus 8.1 bleibt überlegen, da er mehrere Angreifer verkräftet.

# Literaturverzeichnis

- [AAF] M. Andrews, B. Awerbuch, A. Fernandez, T. Leighton, Z. Liu und J. Kleinberg, Universal-Stability results and performance bounds for greedy contention-resolution protocols, *Journal of the ACM*, vol 48 (1), pp. 39-69, 2001.
- [AGKM] V. Arya, N. Garg, R. Khandekar, A. Myerson, K. Munagala und V. Pandit, Local search heuristics for  $k$ -median and facility location problems, *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001.
- [AL] B. Awerbuch und T. Leighton, Improved approximation algorithm for the multi-commodity flow problem and local competitive routing in dynamic networks, *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pp. 487-496, 1994.
- [AKO] W. Aiello, E. Kushilevitz, R. Ostrovsky und A. Rosen, Adaptive packet routing for bursty adversarial traffic, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 359-368, 1998.
- [BKKMS] H. Balakrishnan, M. Frans Kaashoek, D. R. Karger, R. Morris, I. Stoica, Looking up data in P2P systems, *Communications of the ACM (CACM)*, 46 (2), pp. 43-48, 2003.
- [BH] K. Bharat und M.R. Henzinger, Improved algorithms for Topic distillation in a hyperlinked environment, *Proceedings of 21st ACM International Conference on Research and Development in Information Retrieval*, pp. 104-111, 1998.
- [BKR] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan und D.P. Williamson, Adversarial queueing theory, *Journal of the ACM*, Vol. 48 (1), pp. 13-38, 2001.
- [BGMZ] A. Broder, S. Glassman, M. Manasse und G. Zweig, Syntactic clustering of the web, *Proceedings of the 6th International World Wide Web Conference* pp. 391-404, 1997.
- [BCFM] A. Broder, M. Charikar, A. Frieze und M. Mitzenmacher, Min-wise independent permutations, *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 327-336, 1998.
- [CJ] D. Chiu und R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, *Computer Networks and ISDN Systems*, (17), pp. 1-14, 1989.

- [CDR] R. Cole, Y. Dodis und T. Roughgarden, How much can taxes help selfish routing?, *Proceedings of the 4th Annual ACM Symposium on Electronic Commerce*, pp. 98-107, 2003.
- [DBCP] M. Degermark, A. Brodnik, S. Carlsson und S. Pink, Small forwarding tables for fast routing lookups, *Proceedings of the ACM SIGCOMM*, pp. 3-14, 1997.
- [D] O. Drobnik, Verteilte Systeme und Telematik 1, Skript, Institut für Informatik, Johann Wolfgang-Goethe Universität Frankfurt, 2003.
- [DP] X. Deng und C. Papadimitriou, On the complexity of cooperative solution concepts, *Math. Oper. Res.*, 19, pp. 257-266, 1994.
- [DPS] X. Deng, C. Papadimitriou und S. Safra, On the complexity of equilibria, *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pp. 67-71, 2002.
- [DKNS] C. Dwork, R. Kumar, M. Naor und D. Sivakumar, Rank aggregation methods for the web, *Proceedings of the 10th International World Wide Web Conference*, pp. 613-622, 2001.
- [EDD] J. Edmonds, S. Datta und P. Dymond, TCP is competitive against a limited adversary, *Proceedings of the 15th ACM Symposium on parallel algorithms and architectures*, pp. 174-183, 2003.
- [EV] C. Estan und G. Varghese, New directions in traffic management and accounting: focusing on the elephants, ignoring the mice, *ACM Transactions on Computer Systems*, 2003.
- [FKFH] U. Faigle, W. Kern, S.P. Fekete und W. Hochstättler, On the complexity of testing membership in the core of min-cost spanning tree games, *Int. Journal of Game Theory*, 26, pp. 361-366, 1997.
- [FKK] U. Faigle, W. Kern und J. Kuipers, Computing the nucleolus of min-cost spanning tree games is *NP*-hard, *Int. Journal of Game Theory*, 27, pp. 443-450, 1998.
- [FP] W. Fang und L. Peterson, Inter-AS traffic patterns and their implications, *Proceedings of IEEE GLOBECOM*, 1999.
- [FKSS] J. Feigenbaum, A. Krishnamurthy, R. Sami und S. Shenker, Hardness results for multicast cost sharing, erscheint in: *Theoretical Computer Science*, 2003.
- [FGHK] A. Fiat, A. Goldberg, J. Hartline und A. Karlin, Competitive generalized auctions, *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pp. 72-81, 2002.
- [FK] L. Fredman und L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms, *Journal of Algorithms* (21), pp. 618-628, 1996.
- [G] D. Gamarnik, Stability of adaptive and non-adaptive packet routing policies in adversarial queueing networks, *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pp. 206-214, 1999.

- [GHKW] A. Goldberg, J. Hartline, A. Karlin und A. Wright, Competitive auctions, *Manuskript*, 2003.
- [GKS] S. Guha, N. Koudas, K. Shim, Data-streams and histograms, *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pp. 471-475, 2001.
- [GY] N. Garg und N. Young, On-line and End-to-End congestion Control, *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 303-312, 2002.
- [H] B. Hajek, Hitting-time and occupation-time bounds implied by drift analysis with applications, *Advances in Applied Probability*, pp. 502-525, 1982.
- [H02] T. H. Haveliwala, Topic-Sensitive PageRank, *Proceedings of the 11th International World Wide Web Conference*, 2002.
- [JV] K. Jain und V. Vazirani, Applications of Approximation to cooperative games, *33rd Annual ACM Symposium on Theory of Computing*, pp. 364-372, 2001.
- [K1] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, *Journal of the ACM*, (46), pp. 604-632, 1999.
- [K2] J.M. Kleinberg, The small-world phenomenon: an algorithmic perspective, *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pp. 163-170, 2000.
- [KKPS] R. Karp, E. Koutsoupias, C. Papadimitriou und S. Shenker, Optimization problems in congestion control, *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pp. 66-74, 2000.
- [KHR] D. Katabi, M. Handley und C. Rohrs, Internet congestion control for high bandwidth-delay product environments, *Proc. of the ACM SIGCOMM*, 2002.
- [KMS] D. Koukopoulos, M. Mavronicolas und P. Spirakis, FIFO is unstable at arbitrarily low rates, *Electronic Colloquium on Computational Complexity (ECCC)*, Technical Report 03-016, 2003.
- [KP] E. Koutsipias und C. Papadimitrou, Worst-case equilibria, *Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science*, pp. 404-413, 1999.
- [LOS] D. Lehmann, L. O'Callaghan und Y. Shoham, Truth revelation in approximately efficient combinatorial auctions, *Journal of the ACM* (49), pp. 577-602, 2002.
- [LMS] M. Luby, M. Mitzenmacher, M.A. Shokrollahi, D.A. Spielman, V. Steman, Practical loss-resilient codes, *Proc. 29th Annual ACM Symp. on Theory of Computing*, pp.150-159, 1997.
- [M] K.G. Murty, On the linear complementarity problem, *Proceedings of the 3rd Symposium on Operations Research*, pp. 425-439, 1978.
- [Ne] A. Neyman, Finitely repeated games with finite automata, *Mathematics of Operations Research* (23), pp. 513-552, 1998.

- [Ni] N. Nisan, Bidding and allocation in combinatorial auctions, *2nd Annual ACM Conference on Electronic Commerce*, 2000.
- [NR] N. Nisan und A. Ronen, Algorithmic mechanism design, *31st Annual ACM Symposium on Theory of Computing*, pp. 129-140, 1999.
- [PY] C. H. Papadimitriou und M. Yannakakis, On complexity as bounded rationality, *26th Annual ACM Symposium on Theory of Computing*, pp. 726-733, 1994.
- [RT] T. Roughgarden und E. Tardos, How bad is selfish routing?, *Journal of the ACM* 49 (2), pp. 236-259, 2002.
- [SS] L.S. Shapley und M. Shubik, On market games, *Journal of Economic Theory*, 1969.
- [Sh] A. Shokrollahi, Raptor codes, preprint 2003.
- [S] B. von Stengel, Computing equilibria for two-person games, chapter 45 in: *Handbook of Game Theory*, vol 3, eds. R.J. Aumann und S. Hart, North-Holland, pp. 1723-1759, 2002.
- [SMLK] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. Frans Kaashoek, F. Dabek und H. Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications", , *IEEE/ACM Transactions on Networking*, Vol. 11, No. 1, pp. 17-32, 2003.
- [T] A. Tanenbaum, *Computer Networks*, Prentice Hall, 4th edition, 2002.
- [W] M. Weinard, The Necessity of Timekeeping in Adversarial Queueing, *Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA)*, pp. 440-451, 2005.
- [YKT] M. Yajnik, J. Kurose, D. Towsley, Packet Loss Correlation in the Mbone multicast network, *IEEE Global Internet Conference*, 1996.
- [ZN] E. Zurel und N. Nisan, An efficient approximate allocation algorithm for combinatorial auctions, *3rd Annual ACM Conference on Electronic Commerce*, 2001.